INDUSTRIAL SAFETY

NLP BASED CHATBOT

CAPSTONE PROJECT- INTERIM REPORT

SUBMITTED BY

SARAVANAN JANARTHANAN
VIJAY KUMAR VANKADARI
DNYANESHWAR NAGANE
JITENDRA KUMAR
GOPENDRA KUMAR
RAHUL SESHA

PGP-AIML GROUP 8 2021-22

PROJECT MENTOR : DR. AAMIT LAKHANI

Great Lakes Institute of Management in collaboration with

The University of Texas at Austin

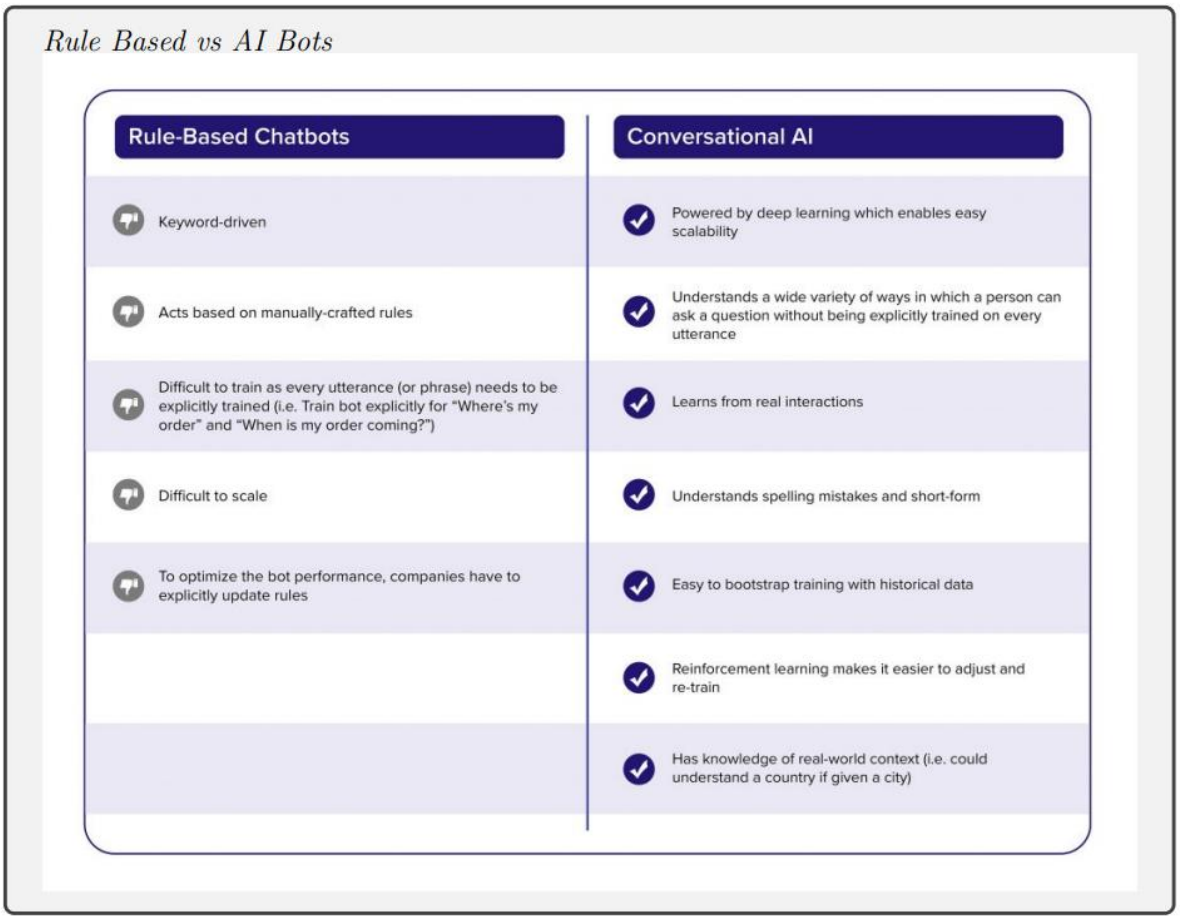
TABLE OF CONTENTS

1. INTRODUCTION

This capstone project is based on building semi-rule chatbot which would help certain industries to understand on why employees continue to suffer some injuries/accidents in plants and try to improvise on the same. This report gives us more insights based on all given milestones.

1.1 Overview

Chatbots are software that use natural language processing (NLP) to engage in conversations with users. Rule-based chatbots provide answers based on a set of if/then rules that can vary in complexity. These rules are defined and implemented by a chatbot designer. At this point, it's worth adding that rule-based chatbots don't understand the context of the conversation. They provide matching answers only when a user uses a keyword or a command they were programmed to answer.



2. EDA & BUSINESS IMPLICATIONS

Variable	Description
Data	timestamp or time/date information
Countries	which country the accident occurred (anonymised)
Local	the city where the manufacturing plant is located (anonymised)
Industry sector	which sector the plant belongs to
Accident level	from I to VI, it registers how severe was the accident (I means not severe but VI means very severe)
Potential Accident Level	Depending on the Accident Level, the database also registers how severe the accident could have been (due to other factors
Genre	if the person is male of female
Employee or Third Party	if the injured person is an employee or a third party
Critical Risk	some description of the risk involved in the accident
Description	Detailed description of how the accident happened

Table 1: Dataset attributes

Checking the first 5 rows and columns of the dataset

	Unnamed: 0	Data	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Genre	Employee or Third Party	Critical Risk	Description
0	0	2016-01-01 00:00:00	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
1	1	2016-01-02 00:00:00	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
2	2	2016-01-06 00:00:00	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...
3	3	2016-01-08 00:00:00	Country_01	Local_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...
4	4	2016-01-10 00:00:00	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...

Fig 1: The dataset

Checking Dataset Information

```
# Checking the dataset info
cap_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 425 entries, 0 to 424
Data columns (total 11 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Unnamed: 0                425 non-null    int64 
 1   Data                      425 non-null    object
 2   Countries                 425 non-null    object
 3   Local                     425 non-null    object
 4   Industry Sector           425 non-null    object
 5   Accident Level            425 non-null    object
 6   Potential Accident Level  425 non-null    object
 7   Genre                     425 non-null    object
 8   Employee or Third Party   425 non-null    object
 9   Critical Risk             425 non-null    object
 10  Description               425 non-null    object
dtypes: int64(1), object(10)
memory usage: 36.6+ KB
```

Fig 2: Checking the dataset info

Fig 2 - Observations:

1) There are about 425 rows and 11 columns in the dataset.
2) Column Description alone contains the text data
3) Only the Unnamed:0 column dtype is integer i.e. int64, rest of the column dtype is object
4) There are no missing values, which we can also confirm via the isnull/isna function going forward
5) We noticed that except a 'date' column all other columns are categorical columns.

2. DATA CLEANSING

2.1 . Removing 'Unnamed' and Renaming 'Data', 'Countries', 'Genre', 'Employee or Third Party' columns

```
# Remove 'Unnamed: 0' column from Data frame
data.drop("Unnamed: 0", axis=1, inplace=True)

# Rename 'Data', 'Countries', 'Genre', 'Employee or Third Party' columns in Data frame
data.rename(columns={'Data':'Date', 'Countries':'Country', 'Genre':'Gender', 'Employee or Third Party':'E

# Get the top 2 rows
data.head(2)
```

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description
0	2016-01-01 00:00:00	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
1	2016-01-02 00:00:00	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...

2.2 Checking describe function to see if we can pick up any other insights.

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Unnamed: 0	425.0	NaN	NaN	NaN	224.084706	125.526786	0.0	118.0	226.0	332.0	438.0
Data	425	287	2017-02-08 00:00:00	6	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Countries	425	3	Country_01	251	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Local	425	12	Local_03	90	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Industry Sector	425	3	Mining	241	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Accident Level	425	5	I	316	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Potential Accident Level	425	6	IV	143	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Genre	425	2	Male	403	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Employee or Third Party	425	3	Third Party	189	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Critical Risk	425	33	Others	232	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Description	425	411	During the activity of chuteo of ore in hopper...	3	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Fig 2.2

Fig 2.2 - Observations:

1) There are no missing values - which we have also noted above

2) Countries - There are 3 unique countries represented in the data, for which Country_01 has the highest i.e. ~ 60%

3) Cities- Across the three countries there are 12 local locations, represented in the data for which 'Local_03' city has the highest frequency i.e. little over 20%

4) Industry Sector - There are 3 industry sectors , with mining carrying the highest representation i.e. over 50%

5) Accident level - though there are 6 categories as per the meta data i.e. 01 to 06, and 01 being lowest severity, and 6 being highest sverity; only 01 to 05 categories are represented in the dataset, indicating that there were no incidents where highest severity of 6 (VI) was witnessed. To add, the highest representation is from severity level 01 (l), representing ~ 74% of the data.

6) Potential Accident level- As noted in the metadata, for each incident the potential accident level is noted, and there are 6 categories, i.e. 01 to 06, and 01 being lowest severity, and 6 being highest severity. Accident severity of 4 (lV) carries the largest representation i.e. 34%

7) Genre/Gender - The data has significantly more representation from Male than Female i.e. 95%

8) Employee/Third party - There are 3 categories, with third party category having the highest representation

9) Critical Risk - There are 33 types of critical risk, with other carrying the highest representation of ~ 55%

10) Description - It appears that there are 411 unique description of the incident / accident, and activity description on the lines of 'During the activity of chuteo of ore in hopper....' seems to have the highest occurence of 3.

2.3 Checking if there are any duplicates by all columns except unnamed.

	Unnamed: 0	Data	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Genre	Employee or Third Party	Critical Risk	Description
76	88	2016-04-01 00:00:00	Country_01	Local_01	Mining	I	V	Male	Third Party (Remote)	Others	In circumstances that two workers of the Abrat...
77	89	2016-04-01 00:00:00	Country_01	Local_01	Mining	I	V	Male	Third Party (Remote)	Others	In circumstances that two workers of the Abrat...
261	275	2016-12-01 00:00:00	Country_01	Local_03	Mining	I	IV	Male	Employee	Others	During the activity of chuteo of ore in hopper...
262	276	2016-12-01 00:00:00	Country_01	Local_03	Mining	I	IV	Male	Employee	Others	During the activity of chuteo of ore in

Fig 2.3 Checking the duplicates:

Fig 2.3 Observations:

1. We note that out of the 26 records, there are 13 records where all the input values are the same for all columns except the Unnamed column
2. While it is not entirely clear whether or not these are duplicates, we noted that there are instance where the duplicate record are in sequence, indicating that perhaps the entry may have been made twice due to oversight / human error
3. All in there are, 6 unique records having 7 duplicates values in total i.e. 2% of the data
4. While we will keep the duplicates for now, during the model building phase we will test whether or not we should drop them, as due to the low no. of records, it could potentially serve the same purpose as synthetic data and help create a more robust model.

2.4 Checking unique values

```
# Check unique values of all columns except 'Description' column
for x in data.columns:
    if x != 'Description':
      print('--'*30); print(f'Unique values of "{x}" column'); print('--'*30)
      print(data[x].unique())
      print('\n')
```

```
------------------------------------------------------------
Unique values of "Date" column
------------------------------------------------------------
['2016-01-01 00:00:00' '2016-01-02 00:00:00' '2016-01-06 00:00:00'
 '2016-01-08 00:00:00' '2016-01-10 00:00:00' '2016-01-12 00:00:00'
 '2016-01-16 00:00:00' '2016-01-17 00:00:00' '2016-01-19 00:00:00'
 '2016-01-26 00:00:00' '2016-01-28 00:00:00' '2016-01-30 00:00:00'
 '2016-02-01 00:00:00' '2016-02-02 00:00:00' '2016-02-04 00:00:00'
 '2016-02-06 00:00:00' '2016-02-07 00:00:00' '2016-02-08 00:00:00'
 '2016-02-21 00:00:00' '2016-02-25 00:00:00' '2016-02-09 00:00:00'
 '2016-02-10 00:00:00' '2016-02-15 00:00:00' '2016-02-14 00:00:00'
 '2016-02-13 00:00:00' '2016-02-16 00:00:00' '2016-02-17 00:00:00'
 '2016-02-19 00:00:00' '2016-02-20 00:00:00' '2016-02-18 00:00:00']


------------------------------------------------------------
Unique values of "Country" column
------------------------------------------------------------
['Country_01' 'Country_02' 'Country_03']


------------------------------------------------------------
Unique values of "Local" column
------------------------------------------------------------
['Local_01' 'Local_02' 'Local_03' 'Local_04' 'Local_05' 'Local_06'
 'Local_07' 'Local_08' 'Local_10' 'Local_09' 'Local_11' 'Local_12']


------------------------------------------------------------
Unique values of "Industry Sector" column
------------------------------------------------------------
['Mining' 'Metals' 'Others']
```

Fig 2.4

Fig 2.4 Observations:

1. We observed that there are records of accidents from 1st Jan 2016 to 9th July 2017 in every month. So there are no outliers in the 'Date' column.
2. There are only three country types so there are no outliers in 'Country' column.
3. There are 12 Local cities where manufacturing plant is located and it's types are in sequence so there are no outliers in 'Local' column.
4. There are only three Industry Sector types which are in sequence so there are no outliers in 'Industry Sector' column.
5. There are only five Accident Level types which are in sequence so there are no outliers in 'Accident Level' column.
6. There are only six Potential Accident Level types which are in sequence so there are no outliers in 'Potential Accident Level' column.
7. There are only two Gender types in the provided data so there are no outliers in 'Gender' column.
8. There are only three Employee types in the provided data so there are no outliers in 'Gender' column.
9. There are quite a lot of Critical risk description and we don't see any outliers but with the help of SME we can decide whether this column has outliers or not.

2.5 Creating New features from 'Date' feature to capture informative information

```
data['Date'] = pd.to_datetime(data['Date'])

data['Year'] = data.Date.apply(lambda x : x.year)
data['Month'] = data.Date.apply(lambda x : x.month)
data['Day'] = data.Date.apply(lambda x : x.day)
data['Weekday'] = data.Date.apply(lambda x : x.day_name())
data['WeekofYear'] = data.Date.apply(lambda x : x.weekofyear)

data.head()
```

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description	Year	Month	Day
0	2016-01-01	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...	2016	1	1
1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	1	2
2	2016-01-06	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	2016	1	6
3	2016-01-08	Country_01	Local_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...	2016	1	8
4	2016-01-10	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...	2016	1	10

Fig 2.5

As we know, this database comes from one of the biggest industry in Brazil which has four climatological seasons as below.

https://seasonsyear.com/Brazil

Spring: September to November

Summer: December to February

Autumn: March to May

Winter: June to August

We can create seasonal variable based on month variable

2.6 Creating the month variable into seasons

```
# Creating the month variable into seasons
def month2seasons(x):
    if x in [9, 10, 11]:
        season = 'Spring'
    elif x in [12, 1, 2]:
        season = 'Summer'
    elif x in [3, 4, 5]:
        season = 'Autumn'
    elif x in [6, 7, 8]:
        season = 'Winter'
    return season
```

```
data['Season'] = data['Month'].apply(month2seasons)
data.head(3)
```

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description	Year	Month	Day	
0	2016-01-01	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08	2016	1	1	
1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	1	2	
2	2016-01-06	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	2016	1	6	W

Fig 2.6

2.7 Creating Holiday Variable

We can create holidays variable based on Brazil holidays list from 2016 and 2017

```
import holidays

brazil_holidays = []

print('--'*40); print('List of Brazil holidays in 2016'); print('--'*40)
for date in holidays.Brazil(years = 2016).items():
    brazil_holidays.append(str(date[0]))
    print(date)

print('--'*40); print('List of Brazil holidays in 2017'); print('--'*40)
for date in holidays.Brazil(years = 2017).items():
    brazil_holidays.append(str(date[0]))
    print(date)
```

```
--------------------------------------------------------------------------------
List of Brazil holidays in 2016
--------------------------------------------------------------------------------
(datetime.date(2016, 1, 1), 'Ano novo')
(datetime.date(2016, 4, 21), 'Tiradentes')
(datetime.date(2016, 5, 1), 'Dia Mundial do Trabalho')
(datetime.date(2016, 9, 7), 'Independência do Brasil')
(datetime.date(2016, 10, 12), 'Nossa Senhora Aparecida')
(datetime.date(2016, 11, 15), 'Proclamação da República')
(datetime.date(2016, 12, 25), 'Natal')
(datetime.date(2016, 3, 25), 'Sexta-feira Santa')
(datetime.date(2016, 3, 27), 'Páscoa')
(datetime.date(2016, 5, 26), 'Corpus Christi')
(datetime.date(2016, 2, 10), 'Quarta-feira de cinzas (Início da Quaresma)')
(datetime.date(2016, 2, 9), 'Carnaval')
--------------------------------------------------------------------------------
List of Brazil holidays in 2017
--------------------------------------------------------------------------------
(datetime.date(2017, 1, 1), 'Ano novo')
(datetime.date(2017, 4, 21), 'Tiradentes')
(datetime.date(2017, 5, 1), 'Dia Mundial do Trabalho')
(datetime.date(2017, 9, 7), 'Independência do Brasil')
(datetime.date(2017, 10, 12), 'Nossa Senhora Aparecida')
(datetime.date(2017, 11, 2), 'Finados')
(datetime.date(2017, 11, 15), 'Proclamação da República')
(datetime.date(2017, 12, 25), 'Natal')
(datetime.date(2017, 4, 14), 'Sexta-feira Santa')
(datetime.date(2017, 4, 16), 'Páscoa')
(datetime.date(2017, 6, 15), 'Corpus Christi')
(datetime.date(2017, 3, 1), 'Quarta-feira de cinzas (Início da Quaresma)')
(datetime.date(2017, 2, 28), 'Carnaval')
```

Fig 2.7

2.8 Creating additional column as "Is_Holiday"

```
#Creating additional column "Is_Holiday" in the data

data['Is_Holiday'] = [1 if str(val).split()[0] in brazil_holidays else 0 for val in data['Date']]
data.head(3)
```

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description	Year	Month	Day	
0	2016-01-01	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...	2016	1	1	
1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	1	2	
2	2016-01-06	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	2016	1	6	W

Fig 2.8

2.9 Univariate Analysis:

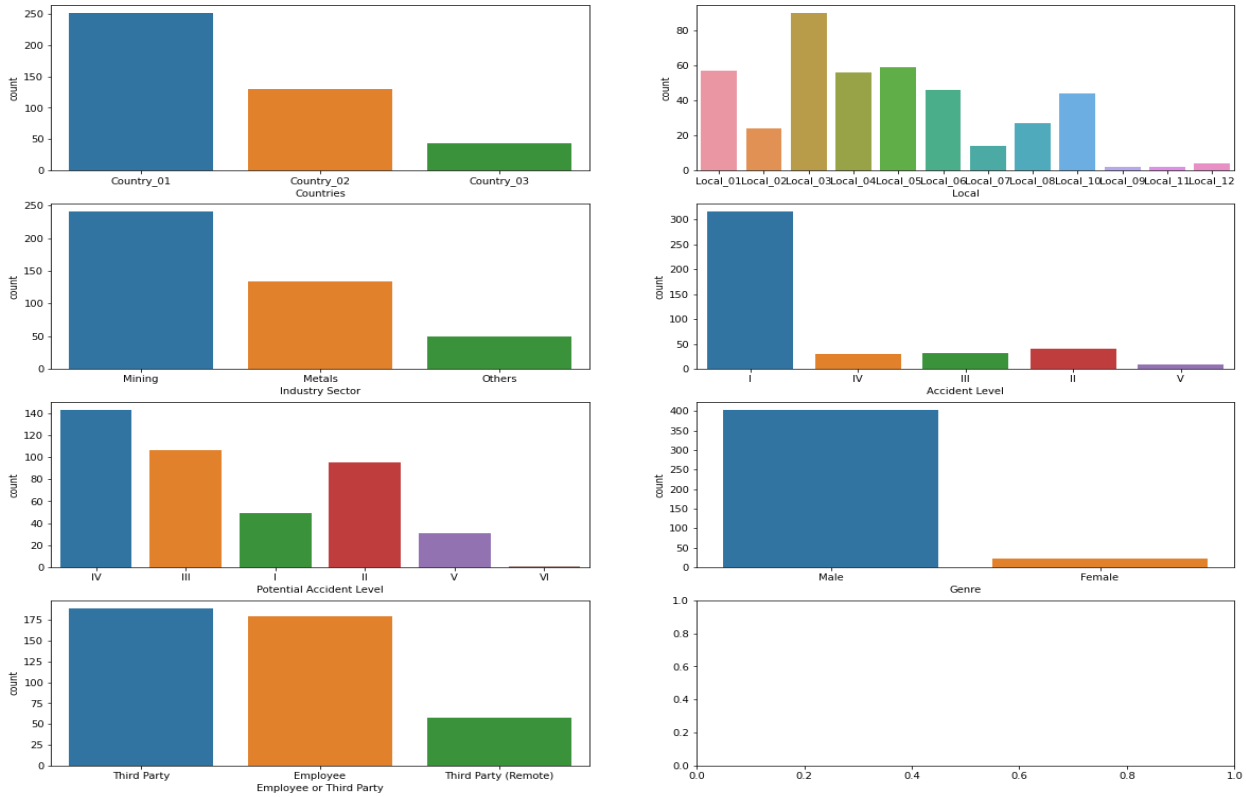


Fig 2.9

Observations:

1) Country - Country 1 has the highest representation, country 3 has the lowest representation.

2) City/Location - Location_03 has the highest instance of records, and location_10 & location_11 have the lowest instance

3) Industry - Mining has the highest instances of records, Other has the lowest instance of records - it is not clear which other sectors are included

4) Accident Level (severity) - 1 has the highest instance of accident levels; 5 has the lowest levels

5) Potential accident level - 4 has the highest instance of accident level; 6 has the lowest instance of accident levels

6) Gender - The Female gender representation is meagre

7) Employee or Third party - The 'third party' category has the highest instance of employee type category, and third party remote has the lowest instance of category

2.10 Analysing Critical Risk

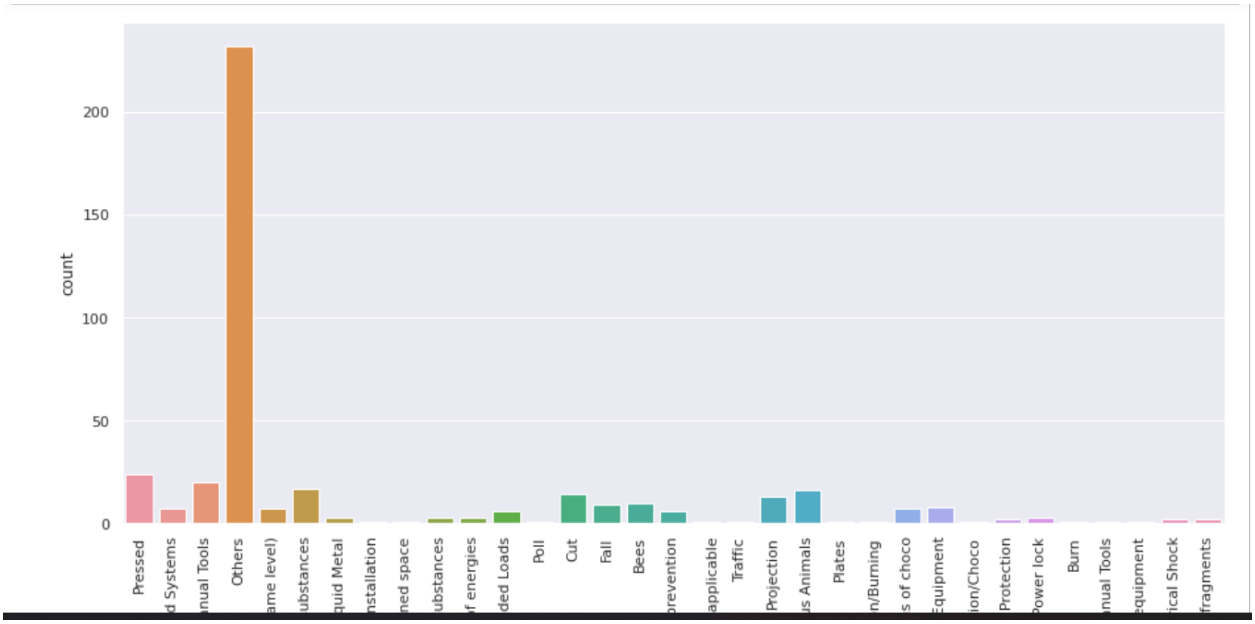


Fig 2.10

Observations :

1. Others has the highest instance of critical risk category; far behind though relatively significant other critical risk categories are - Pressed, Manual tools, chemical substance, venomous animals.
2. Lowest / negligible categories in terms of representation are Burn, Projection/Manual tools, Projection/Choco, Plates, plates, poll, traffic, not applicable and Individual projection equipment.c1

2.11 Finding top 10 Critical Risk factors

```
# Finding top 10 Critical Risk factors

top10_criticalrisk=crticalrisk_df[0:10]
top10_criticalrisk
```

	Critical Risk
Others	232
Pressed	24
Manual Tools	20
Chemical substances	17
Venomous Animals	16
Cut	14
Projection	13
Bees	10
Fall	9
Vehicles and Mobile Equipment	8

```
# We can see that the bottom categories have only 1 instance,and given these might not be very useful fo
# or combine these categories with other
# Even the 11th value from the last one, i.e. not applicable though not shown here carries only 1 record
# in terms of the bottom10 represented categories. To add, any representation less than 5 can be combine
# predict accurately, and it's best to jointly represent them in the others category, as these are or le
# impact the use case.

bottom10_criticalrisk=crticalrisk_df.tail(10)
bottom10_criticalrisk
```

	Critical Risk
Projection/Manual Tools	1
Burn	1
Poll	1
Projection/Choco	1
Projection/Burning	1
Plates	1
Confined space	1
Traffic	1
\nNot applicable	1
Electrical installation	1

Fig 2.11

2.12 Taking the cross tab/pivot analysis of potential accident level against Accident level

Potential Accident Level \ Accident Level	I	II	III	IV	V	Data All
I	49.0	NaN	NaN	NaN	NaN	49
II	88.0	7.0	NaN	NaN	NaN	95
III	89.0	14.0	3.0	NaN	NaN	106
IV	80.0	16.0	26.0	21.0	NaN	143
V	10.0	3.0	2.0	9.0	7.0	31
VI	NaN	NaN	NaN	NaN	1.0	1
All	316.0	40.0	31.0	30.0	8.0	425

Fig 2.12

Observations:

1) Severity 1 - All potential accident severity levels were matched with the actual accident severity levels i.e. 49 instances

2) Severity 2 - Of the 95 instances marked as 2 severity in terms of potential accident levels, a majority of them (i.e. 93%) were actually falling in the 1 severity category

3) Severity 3- of the 106 potential accident level incidents at severity 3, only 3 instances were actually graded as severity 3 (i.e. ~ 3%)

4) Severity 4 - of the total 143 potentially accident levels, only 21 (i.e. ~15%), and a majority were actually of 1 severity levels (i.e. > 50%) and the remaining at severity level 3 and 2 in decreasing order.

5) Severity 5 - Of the 31 potential accident levels at severity 5, only 7 (i.e.23%) were actually graded as 5 severity level. A majority of them were graded at actual severity levels of 1 and 4, representing 30-35% each.

6) Severity 6- The severity instance being only 1 in terms of potential severity levels, was actually classified as 5.

7) **Conclusion -** It is best to consider the Actual severity levels, as:

   i) they were based on actual / factual events, rather than potential which might have been derived by experts and/or actual in the past, and due to the improvement levels of the safety measures, modernized equipment and leveraging new technology, the actual severity might be lower.

   ii) including both potential and actual severity, might make the model confused, and unfavourably impact the accuracy levels. Another point being on the potential accuracy levels for each instance might be unique and discretely imputed by the user/recorder which might be incorrect in terms of input data.

2.13 Country level representation against location

```
# Let's look at the country level representation against location

country_table = pd.pivot_table(cap_data, values=['Genre'],index=['Countries'], columns=['Local'],
                               aggfunc=np.count_nonzero, margins=True)
```

```
# We note that each location is unique for each country i.e. local_appearing for country_01, will have n
# As the data is more biased towards country_01 (~ 60%), i.e. it does not make sense to split the data b
# due to the fact that it will impact the accuracy of predicting actual severity for country_02 and coun

country_table
```

Local	Local_01	Local_02	Local_03	Local_04	Local_05	Local_06	Local_07	Local_08	Local_09	Local_10	Local_11	Lo
Countries												
Country_01	57.0	NaN	90.0	56.0	NaN	46.0	NaN	NaN	NaN	NaN	2.0	
Country_02	NaN	24.0	NaN	NaN	59.0	NaN	14.0	27.0	2.0	NaN	NaN	
Country_03	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	44.0	NaN	
All	57.0	24.0	90.0	56.0	59.0	46.0	14.0	27.0	2.0	44.0	2.0	

Fig 2.13

2.14 Dropping Unnamed column

```
# Along with the Countries, Local,Potential Accident Level columns- we will also drop the Unnamed column
cap_data1=cap_data.drop(columns=['Countries', 'Local','Potential Accident Level','Unnamed: 0'],axis=1)
```

```
cap_data1.head()
```

	Data	Industry Sector	Accident Level	Genre	Employee or Third Party	Critical Risk	Description
0	2016-01-01 00:00:00	Mining	I	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
1	2016-01-02 00:00:00	Mining	I	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
2	2016-01-06 00:00:00	Mining	I	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...
3	2016-01-08 00:00:00	Mining	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...
4	2016-01-10 00:00:00	Mining	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...

```
cap_data1.columns
```

```
Index(['Data', 'Industry Sector', 'Accident Level', 'Genre',
       'Employee or Third Party', 'Critical Risk', 'Description'],
      dtype='object')
```

```
cap_data1.Data
```

```
0      2016-01-01 00:00:00
1      2016-01-02 00:00:00
2      2016-01-06 00:00:00
3      2016-01-08 00:00:00
4      2016-01-10 00:00:00
               ...
420    2017-07-04 00:00:00
421    2017-07-04 00:00:00
422    2017-07-05 00:00:00
423    2017-07-06 00:00:00
424    2017-07-09 00:00:00
Name: Data, Length: 425, dtype: object
```

Fig 2.14

2.15 Rearranging the columns and dropping the 'data' column

```
# let's rearrange the columns and drop the 'data' column, as the information has been extracted in terms
cap_data1.head()
```

	Data	Industry Sector	Accident Level	Genre	Employee or Third Party	Critical Risk	Description	day	month	year
0	2016-01-01	Mining	I	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...	1	1	2016
1	2016-01-02	Mining	I	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2	1	2016
2	2016-01-06	Mining	I	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	6	1	2016
3	2016-01-08	Mining	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...	8	1	2016
4	2016-01-10	Mining	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...	10	1	2016

```
cap_data1.drop(columns='Data',axis=1,inplace=True)
```

```
cap_data1.columns
```

```
Index(['Industry Sector', 'Accident Level', 'Genre', 'Employee or Third Party',
       'Critical Risk', 'Description', 'day', 'month', 'year'],
      dtype='object')
```

```
cap_data1=cap_data1[['day', 'month', 'year','Industry Sector', 'Accident Level', 'Genre', 'Employee or T
```

```
cap_data1.head()
```

	day	month	year	Industry Sector	Accident Level	Genre	Employee or Third Party	Critical Risk	Description
0	1	1	2016	Mining	I	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
1	2	1	2016	Mining	I	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
2	6	1	2016	Mining	I	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...
3	8	1	2016	Mining	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...
4	10	1	2016	Mining	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...

Fig 2.15

2.16 Taking year value count

```
# Taking year value count
cap_data1.year.value_counts()
```

```
2016    285
2017    140
Name: year, dtype: int64
```

- While the data is split between 2016 and 2017, with majority coming from 2016, the year column is not likely to add any value to the model training can can be dropped

Fig 2.16

2.17 Checking the impact of incidents by month

```
# let's look at the impact of incidents by month to see if there are any insights

cap_data1.month.value_counts().sort_values(ascending=False)
```

```
2     61
3     53
4     52
6     51
5     41
1     40
7     24
9     24
12    24
8     21
10    21
11    13
Name: month, dtype: int64
```

Inference:

- It seems like majority of accidents have happened during the first 6 months of the year, indicating potential industry season, due to specific reasons, for instance favourable external/weather conditions.

Fig 2.17

2.18 Checking actual severity by month

```
# let's look at the actual severity by month, to see if there are any insights

month_table = pd.pivot_table(cap_data1, values=['Genre'],index=['month'], columns=['Accident Level'],
                             aggfunc=np.count_nonzero, margins=True)
month_table
```

	Genre					
Accident Level	I	II	III	IV	V	All
month						
1	33.0	2.0	2.0	2.0	1.0	40
2	42.0	9.0	4.0	5.0	1.0	61
3	37.0	7.0	3.0	3.0	3.0	53
4	44.0	2.0	3.0	3.0	NaN	52
5	32.0	3.0	1.0	4.0	1.0	41
6	41.0	3.0	2.0	4.0	1.0	51
7	16.0	1.0	4.0	2.0	1.0	24
8	15.0	3.0	2.0	1.0	NaN	21

Fig 2.18

Inference:

1) It appears that there is no particular trend with regards to the month, except that majority of the data is represented by severity 1 for all months
2) severity 2 and 4 were seen highest in Februray, however this is not significantly differnt from March
3) From August onwards there are no severity incidents of level 5 (V)

2.19 dropping month and day

```
# Dropping month and day
cap_data1.drop(columns=['month','day'],axis=1,inplace=True)
```

```
cap_data1.head()
```

	Industry Sector	Accident Level	Genre	Employee or Third Party	Critical Risk	Description
0	Mining	I	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
1	Mining	I	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
2	Mining	I	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...
3	Mining	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...
4	Mining	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...

2.20analysis by gender and accident levels

```
# let's do an analysis by gender and accident levels
gender_table = pd.pivot_table(cap_data1, values=['Critical Risk'],index=['Genre'], columns=['Accident Le
                              aggfunc=np.count_nonzero, margins=True)

gender_table
```

	Critical Risk					
Accident Level	**I**	**II**	**III**	**IV**	**V**	**All**
Genre						
Female	18.0	3.0	1.0	NaN	NaN	22
Male	298.0	37.0	30.0	30.0	8.0	403
All	316.0	40.0	31.0	30.0	8.0	425

Fig 2.20

Inference:

1) From the above we note that there were no severity 4 and 5 incidents occurring for female employees/ staff; indicating that men are preferred for riskeir operations / jobs.
2) All severity 4 & 5 incidents happened for the male gender
3) For the female gender and male gender 82% and 74% of the incidents were at severity level 1
4) For the female gender only 1/22 incident (~ 5%) occured in the severity category 3
5) For the male gender the 2nd highest instances happened at the severity level 2 at 9%, with the same proportion represented in the severity category 3 and 4 i.e. 7% each.

2.21 Checking gender, employee and critical risk pivot

```
# let's look at gender, employee and critical risk pivot
gendernemp_table = pd.pivot_table(cap_data1, values=['Critical Risk'],index=['Genre','Employee or Third
                                  aggfunc=np.count_nonzero, margins=True)

gendernemp_table
```

		Critical Risk					
	Accident Level	I	II	III	IV	V	All
Genre	Employee or Third Party						
Female	Employee	6.0	1.0	1.0	NaN	NaN	8
	Third Party	8.0	1.0	NaN	NaN	NaN	9
	Third Party (Remote)	4.0	1.0	NaN	NaN	NaN	5
Male	Employee	134.0	14.0	13.0	10.0	NaN	171
	Third Party	126.0	18.0	14.0	16.0	6.0	180
	Third Party (Remote)	38.0	5.0	3.0	4.0	2.0	52
All		316.0	40.0	31.0	30.0	8.0	425

Fig 2.21

Inference:

1) For Female gender, the employee or third party type for severity 1 appears to be related i.e. for third party being highest and third party remote being lowest, while for severity 2 it did not make any difference.
2) For Male, the severity level 3 and 4, had higher representation from third party employees compared to Employee which had the same total representation, and for severity 5, there were no instances for the employee and a majority of them coming from third party.

2.22 Adding new dimension and checking total words for each description

```
# let's add a new dimension and check the total of the words for each f the descrioption

cap_data1['len_Description'] = cap_data1['Description'].apply(lambda x: len(x.split(" ")))
cap_data1
```

	Industry Sector	Accident Level	Genre	Employee or Third Party	Critical Risk	Description	len_Description
0	Mining	I	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...	80
1	Mining	I	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	54
2	Mining	I	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	57
3	Mining	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...	97
4	Mining	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...	88
...	...	...	...	...	...	...	...
420	Mining	I	Male	Third Party	Others	Being approximately 5:00 a.m. approximately, w...	38
421	Mining	I	Female	Employee	Others	The collaborator moved from the infrastructure...	39
422	Metals	I	Male	Employee	Venomous Animals	During the environmental monitoring activity i...	44
423	Metals	I	Male	Employee	Cut	The Employee performed the activity of strippi...	33
424	Mining	I	Female	Third Party	Fall prevention (same level)	At 10:00 a.m., when the assistant cleaned the ...	35

425 rows × 7 columns

Fig 2.22

2.23 Describing the data

```
# we can see that the majority of words i.e. 75% of the data is under 85 word limit

# though the hugh difference between the words in the 75% and 100% percentile indicates possible presenc

cap_data1.len_Description.describe()
```

```
count    425.000000
mean      65.661176
std       32.436048
min       16.000000
25%       40.000000
50%       60.000000
75%       84.000000
max      183.000000
Name: len_Description, dtype: float64
```

Fig 2.23

2.24 Graph Description

```
# As noted above a majority of the records have words falling within the 85 word limit, also suggesting
description_words = []

for i in cap_data1['Description']:
  description_words.append(len(i.split()))

word_length = pd.DataFrame({'Description':description_words})
sns.set(rc={'figure.figsize':(15,7)})
word_length.hist(bins = 40)
plt.show()
```

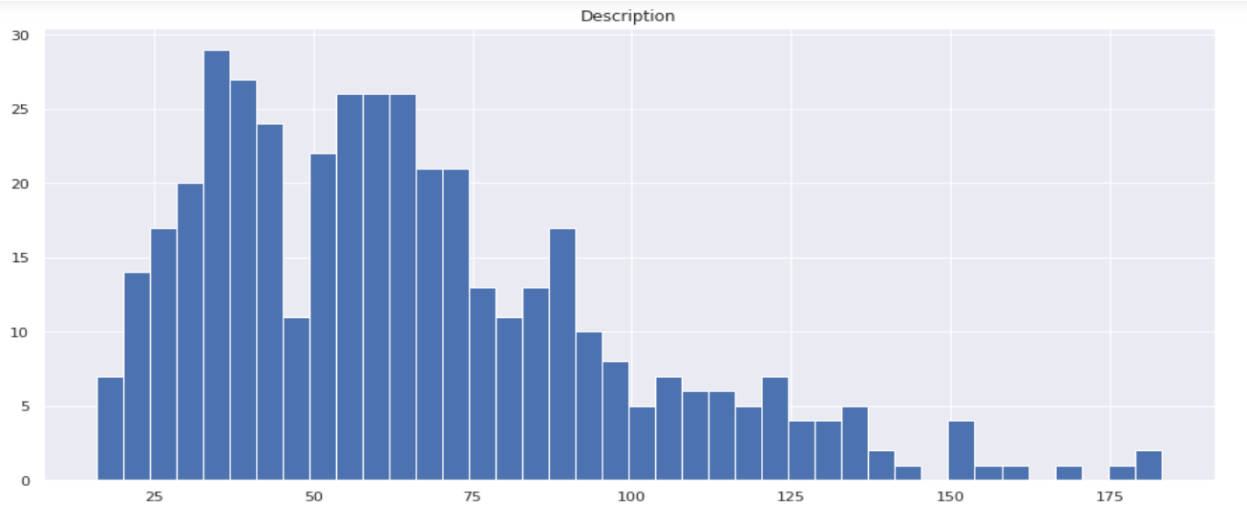


Fig 2.24

1) As the model will be built in such a way that the user will type the free text describing the task, it is recommended not to cut/limit the word text but pad it to say 250 words limit.

3. DATA PREPROCESSING (NLP PREPROCESSING TECHNIQUES)

3.1 Creating a copy of description column

```
# Create a copy of description column so that orginal copy is retained post cleanup
data['desc_copy'] = data['Description']
```

```
data.head()
```

	Date	Country	Local	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description	Year	Month	Day
0	2016-01-01	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...	2016	1	1
1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	1	2
2	2016-01-06	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	2016	1	6

Fig 3.1

```
import re
import nltk
```

```
# reomove any characters that has ascii value greater than 127 for non-English character
data['desc_copy'] = data['desc_copy'].apply(lambda x: ''.join(c for c in x if 0 < ord(c) < 127) )
```

```
# Eliminate All special Characters and numbers
data['desc_copy'] = data['desc_copy'].apply(lambda x: re.sub(r"[^a-zA-Z ]", "", x) )
```

```
# Convert all textual data to lowercase
data['desc_copy'] = data['desc_copy'].apply(lambda x: x.lower() )
```

```
# Remove all Stopwords
from nltk.corpus import stopwords
nltk.download('stopwords')
engStopWords = stopwords.words("english")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\poorn\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

3.3 Creating a copy of 'Potential Accident Level' feature to convert it to categorical values

```
#create a copy of 'Potential Accident Level' feature to convert it to categorical values
data['Potential_Accident_Level_cat'] = data['Potential Accident Level']
```

```
data['Potential_Accident_Level_cat'] = data['Potential_Accident_Level_cat'].astype('category')
```

```
data['Potential_Accident_Level_cat'][:5]
```

```
0     IV
1     IV
2    III
3      I
4     IV
Name: Potential_Accident_Level_cat, dtype: category
Categories (6, object): ['I', 'II', 'III', 'IV', 'V', 'VI']
```

```
data['Potential_Accident_Level_cat'] = data['Potential_Accident_Level_cat'].cat.codes
```

```
data['Potential_Accident_Level_cat'][:5]
```

```
0    3
1    3
2    2
3    0
4    3
Name: Potential_Accident_Level_cat, dtype: int8
```

Fig 3.3

Few of the NLP pre-processing steps taken before applying model on the data

1. Converting to lower case, avoid any capital cases
2. Removing punctuations
3. Lemmatization
4. Removing stop words

```
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
import nltk
nltk.download('all')
nltk.download('stopwords',quiet=True) # stopword library
nltk.download('wordnet', quiet=True) # wordnet library
nltk.download('words', quiet=True) # words library
nltk.download('punkt', quiet=True) # tokenize library
import re


def remove_specialchars_digits(text):
    return re.sub("(\\d|\\W)+"," ", text)

def lc(text):
    return text.lower()

def rem_stopword(text):
    token = nltk.word_tokenize(text)
    text_stop = [x for x in token if x not in set(stopwords.words('english'))]
    lemmatizer = WordNetLemmatizer()
    text_lemma = [lemmatizer.lemmatize(word) for word in text_stop]
    text_lemma = ' '.join(text_lemma)
    return text_lemma

def rem_punct(text):
    return re.sub('[^a-zA-Z]', ' ', str(text))

def rem_tags(text):
    return re.sub("&lt;/?.*?&gt;"," &lt;&gt; ", text)


print('--'*30); print('Converting description to lower case')
data['Cleaned_Description'] = data['Description'].apply(lambda x : x.lower())

print('Removing punctuations')
data['Cleaned_Description'] = data['Cleaned_Description'].apply(lambda x: rem_punct(x))

print('Removing multiple spaces between words')
```

```
data['Cleaned_Description'] = data['Cleaned_Description'].apply(lambda x: rem_tags(x))

print('Removing stop words')
data['Cleaned_Description'] = data['Cleaned_Description'].apply(lambda x: rem_stopword(x))

print('--'*30)
```

```
[nltk_data] Downloading collection 'all'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /root/nltk_data...
[nltk_data]    |   Package abc is already up-to-date!
[nltk_data]    | Downloading package alpino to /root/nltk_data...
[nltk_data]    |   Package alpino is already up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger_ru to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger_ru is already
[nltk_data]    |       up-to-date!
```

3.5 Getting the Length of each line

Get the Length of each line and find the maximum length as different lines are of different length. We need to pad our sequences using the max length.

```
print('--'*45); print('Get the length of each line, find the maximum length and print the maximum length
print('Length of line ranges from 64 to 672.'); print('--'*45)

# Get length of each line
data['line_length'] = data['Cleaned_Description'].str.len()

print('Minimum line length: {}'.format(data['line_length'].min()))
print('Maximum line length: {}'.format(data['line_length'].max()))
print('Line with maximum length: {}'.format(data[data['line_length'] == data['line_length'].max()]['Clea
```

```
------------------------------------------------------------------------------------------
Get the length of each line, find the maximum length and print the maximum length line
Length of line ranges from 64 to 672.
------------------------------------------------------------------------------------------
Minimum line length: 61
Maximum line length: 657
Line with maximum length: level gallery holding activity bolter equipment operator performs drilling
first hole support right gable foot deep drill end drill rod break leaving thread inside drilling ma
chine shank operator assistant decide make two empty percussion attempt free thread shank without su
ccess third attempt assistant enters corrugated iron central hole rest bar embedded shank generate p
ressure moment operator activates percussion generates movement shank hit palm victim left hand gene
rating described injury worker wearing safety glove time accident end corrugated iron contact left h
and shaped like cane worker time accident positioned roof supported mesh split set
```

3.6 Getting the number of words

```
print('--'*45); print('Get the number of words, find the maximum number of words and print the maximum n
print('Number of words ranges from 10 to 98.'); print('--'*45)

# Get length of each line
data['nb_words'] = data['Cleaned_Description'].apply(lambda x: len(x.split(' ')))

print('Minimum number of words: {}'.format(data['nb_words'].min()))
print('Maximum number of words: {}'.format(data['nb_words'].max()))
print('Line with maximum number of words: {}'.format(data[data['nb_words'] == data['nb_words'].max()]['C
```

```
------------------------------------------------------------------------------------------
Get the number of words, find the maximum number of words and print the maximum number of words
Number of words ranges from 10 to 98.
------------------------------------------------------------------------------------------
Minimum number of words: 9
Maximum number of words: 95
Line with maximum number of words: performing sleeve removal maneuver hole meter deep general da sil
va pressed one side locking nut rod together jack hold entire weight rod maneuver locking procedure
effective weight rod secured steel wire rope probe winch moment driller pedro released brake winch i
nefficacy locking done one side chestnut without aid monkey caused sliding rod auxiliary prepared ma
nual unlocking rod holding faucet key firmly probe tower composition shifted stem slid hand shifted
downward causing left hand strike base probe tower structure causing cut th th quirodactyl employee
taken hospital went medical care wound sutured stitch removed day activity
```

Fig 3.6

3.7 Checking different body-related, employee related, movement-related, equipment-related and accident-related words.

```
from wordcloud import WordCloud

wordcloud = WordCloud(width = 1500, height = 800, random_state=0, background_color='black', colormap='ra
                      min_font_size=5, max_words=300, collocations=False).generate(" ".join(data['Cleane
plt.figure(figsize=(15,10))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```

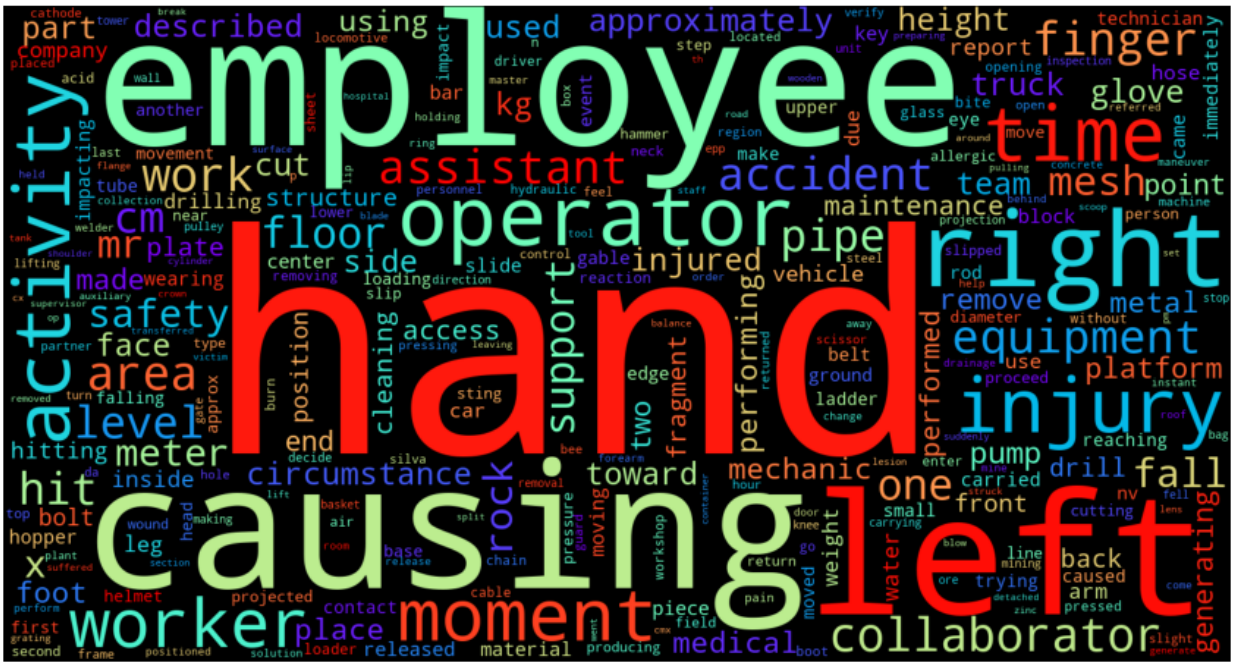


Fig 3.7

Observations

There are many body-related, employee related, movement-related, equipment-related and accident-related words.

- Body-related: left, right, hand, finger, face, foot and glove, arm, leg, head, eye etc.
- Employee-related: employee, operator, collaborator, assistant, worker and mechanic
- Movement-related: fall, hit, lift and slip
- Equipment-related: equipment, pump, meter, drill, truck and tube
- Accident-related: accident, activity, safety, injury, causing, hitting drilling, impacting.

3.8 NLP text summary statistics

```
print('--'*30); print('Five point summary for number of words')
display(data['nb_words'].describe().round(0).astype(int));

print('99% quantilie: {}'.format(data['nb_words'].quantile(0.99)));print('--'*30)
```

```
------------------------------------------------------------
Five point summary for number of words

count    418
mean      33
std       16
min        9
25%       21
50%       30
75%       41
max       95
Name: nb_words, dtype: int64

99% quantilie: 77.82999999999998
```

NLP Pre-processing Summary:

- 74% of data where accident description > 100 is captured in low accident level.
- 34% of data where accident description > 100 is captured in high medium potential accident level.
- 25% of data where accident description > 100 is captured in medium potential accident level.
- 23% of data where accident description > 100 is captured in low potential accident level.
- Few of the NLP pre-processing steps taken before applying model on the data
    - Converting to lower case, avoid any capital cases
    - Converting apostrophe to the standard lexicons
    - Removing punctuations
    - Lemmatization
    - Removing stop words
    - After pre-processing steps:
- Minimum line length: 64
- Maximum line length: 672
- Minimum number of words: 10
- Maximum number of words: 98

4. DATA PREPARATION

4.1 Variable Creation - Word2Vec Embeddings

```
from gensim.models import Word2Vec
from keras.models import Model
from keras.models import load_model

#!pip install --upgrade gensim

import gensim
print(gensim.__version__)

# define training data
sentences = data['Cleaned_Description']

# train model
model = Word2Vec(sentences, min_count=1)

# summarize the loaded model
print(model)
```

```
# summarize vocabulary
words = list(model.wv.index_to_key)
print(words)

# save model
model.save('model.bin')

# load model
new_model = Word2Vec.load('model.bin')
print(new_model)
```

```
WARNING:gensim.models.word2vec:Each 'sentences' item should be a list of words (usually unicode stri
ngs). First item here is instead plain <class 'str'>.
```

```
4.2.0
Word2Vec<vocab=27, vector_size=100, alpha=0.025>
[' ', 'e', 'i', 'r', 't', 'a', 'n', 'o', 'l', 'c', 's', 'd', 'p', 'm', 'g', 'u', 'h', 'f', 'y', 'b',
'v', 'k', 'w', 'x', 'j', 'q', 'z']
Word2Vec<vocab=27, vector_size=100, alpha=0.025>
```

4.2 Variable Creation - Glove Word Embeddings

```
embeddings_index = {}
EMBEDDING_FILE = '/content/drive/MyDrive/CapstoneProject/glove.6B.200d.txt'
f = open(EMBEDDING_FILE)
for line in tqdm(f):
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))
```

```
400000it [01:00, 6585.95it/s]
```

```
Found 400000 word vectors.
```

4.3 Creating normalized vector for whole sentence

```
# this function creates a normalized vector for the whole sentence
def sent2vec(s):
    words = str(s).lower()
    words = word_tokenize(words)
    words = [w for w in words if not w in stop_words]
    words = [w for w in words if w.isalpha()]
    M = []
    for w in words:
        try:
            M.append(embeddings_index[w])
        except:
            continue
    M = np.array(M)
    v = M.sum(axis=0)
    if type(v) != np.ndarray:
        return np.zeros(300)
    return v / np.sqrt((v ** 2).sum())
```

4.4 Variable Creation - TFIDF Features

```
ind_tfidf_df = pd.DataFrame()
for i in [1,2,3]:
    vec_tfidf = TfidfVectorizer(max_features=10, norm='l2', stop_words='english', lowercase=True, use_id
    X = vec_tfidf.fit_transform(data['Cleaned_Description']).toarray()
    tfs = pd.DataFrame(X, columns=["TFIDF_" + n for n in vec_tfidf.get_feature_names()])
    ind_tfidf_df = pd.concat([ind_tfidf_df.reset_index(drop=True), tfs.reset_index(drop=True)], axis=1)

ind_tfidf_df.head(3)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_
feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Plea
se use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

	TFIDF_activity	TFIDF_causing	TFIDF_employee	TFIDF_hand	TFIDF_injury	TFIDF_left	TFIDF_moment	TFIDF_operator	TFID
0	0.0	0.000000	0.0	0.644279	0.000000	0.00000	0.764791	0.0	
1	0.0	0.000000	0.0	0.000000	0.000000	0.00000	0.000000	0.0	

4.5 Variable Creation - Label Encoding

```
# To replace white space everywhere in Employee type
data['Employee type'] = data['Employee type'].str.replace(' ', '_')
data['Employee type'].value_counts()
```

```
Third_Party             185
Employee                178
Third_Party_(Remote)     55
Name: Employee type, dtype: int64
```

```
# To replace white space everywhere in Critical Risk
data['Critical Risk'] = data['Critical Risk'].str.replace('\n', '').str.replace(' ', '_')
data['Critical Risk'].value_counts().head()
```

```
Others                 229
Pressed                 24
Manual_Tools            20
Chemical_substances     17
Cut                     14
Name: Critical Risk, dtype: int64
```

4.6 Sampling Techniques - Create Training and Test Set

```
ind_feat_df = ind_feat_df.append(ind_feat_df[ind_feat_df['Potential Accident Level'] == 5], ignore_index

X = ind_feat_df.drop(['Potential Accident Level'], axis = 1) # Considering all Predictors
y = ind_feat_df['Potential Accident Level']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 1, stratify =

print('X_train shape : ({0},{1})'.format(X_train.shape[0], X_train.shape[1]))
print('y_train shape : ({0},)'.format(y_train.shape[0]))
print('X_test shape : ({0},{1})'.format(X_test.shape[0], X_test.shape[1]))
print('y_test shape : ({0},)'.format(y_test.shape[0]))
```

```
X_train shape : (335,87)
y_train shape : (335,)
X_test shape : (84,87)
y_test shape : (84,)
```

4.7 Resampling Techniques — Oversample minority class

```
ind_feat_df['Potential Accident Level'].value_counts()
```

```
3    141
2    106
1     95
0     45
4     30
5      2
Name: Potential Accident Level, dtype: int64
```

```
# Concatenate our training data back together
X_up = pd.concat([X_train, y_train], axis=1)

# Get the majority and minority class
acclevel_3_majority = X_up[X_up['Potential Accident Level'] == 3]
acclevel_0_minority = X_up[X_up['Potential Accident Level'] == 0]
acclevel_1_minority = X_up[X_up['Potential Accident Level'] == 1]
acclevel_2_minority = X_up[X_up['Potential Accident Level'] == 2]
acclevel_4_minority = X_up[X_up['Potential Accident Level'] == 4]
acclevel_5_minority = X_up[X_up['Potential Accident Level'] == 5]

# Upsample Level0 minority class
acclevel_0_minority_upsampled = resample(acclevel_0_minority,
                                replace = True, # sample with replacement
                                n_samples = len(acclevel_3_majority), # to match majority class
                                random_state = 1)

# Upsample Level1 minority class
acclevel_1_minority_upsampled = resample(acclevel_1_minority,
                                replace = True, # sample with replacement
```

4.8 Varible Tansformation (Normalization and Scaling)

```
# Transform independent features
scaler_X = StandardScaler()#StandardScaler()
pipeline = Pipeline(steps=[('s', scaler_X)])
X_train.iloc[:,:6] = pipeline.fit_transform(X_train.iloc[:,:6]) # Scaling only first 6 feautres

X_test.iloc[:,:6] = pipeline.fit_transform(X_test.iloc[:,:6]) # Scaling only first 6 feautres
```

```
X_train.head(8)
```

	Year	Month	Day	WeekofYear	Season	Weekday	Accident Level	Country_02	Country_03	Local_02	...	TFIDF_ d
161	-0.699206	0.545743	-1.539305	0.361604	0.769691	1.211275	4	0	0	0	...	
395	1.430194	0.231613	-0.726316	0.146695	0.769691	0.668066	0	0	0	0	...	
236	-0.699206	1.488134	-0.261751	1.436153	1.750095	0.124857	1	1	0	1	...	
91	-0.699206	-0.396648	0.086673	-0.426398	-0.210714	1.211275	0	0	0	0	...	
182	-0.699206	0.859873	-1.655446	0.719787	0.769691	-1.504769	1	1	0	0	...	

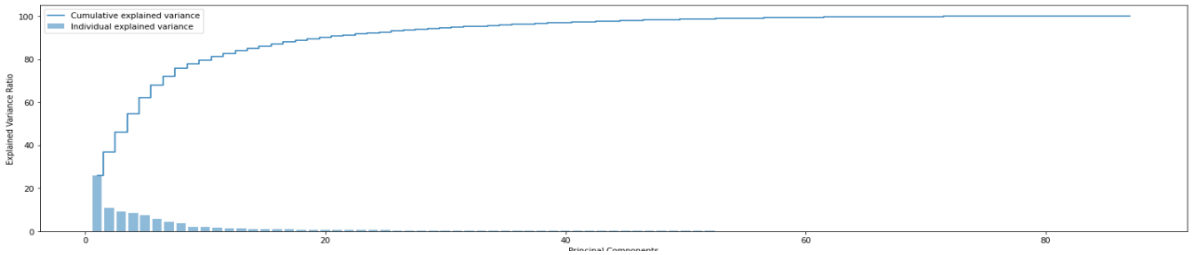
4.9 Use PCA - Extract Principal Components that capture about 95% of the variance in the data

```
# generating the covariance matrix and the eigen values for the PCA analysis
cov_matrix = np.cov(X_train.T) # the relevanat covariance matrix
print('Covariance Matrix \n%s', cov_matrix)

#generating the eigen values and the eigen vectors
e_vals, e_vecs = np.linalg.eig(cov_matrix)
print('Eigenvectors \n%s' %e_vecs)
print('\nEigenvalues \n%s' %e_vals)
```

```
Covariance Matrix
%s [[ 1.00299401e+00 -4.06343355e-01 -1.69641011e-02 ... -6.84270897e-03
  -2.29167927e-03 -3.02316048e-03]
 [-4.06343355e-01  1.00299401e+00  1.86428110e-02 ...  7.38129463e-03
   1.58394798e-03  6.87909156e-03]
 [-1.69641011e-02  1.86428110e-02  1.00299401e+00 ...  6.07768739e-04
  -1.37765560e-02 -8.50002984e-03]
 ...
```

4.10 Plotting the variance explained by the principal components and the cumulative variance

```
plt.figure(figsize=(20 , 5))
plt.bar(range(1, e_vals.size + 1), var_exp, alpha = 0.5, align = 'center', label = 'Individual explained
plt.step(range(1, e_vals.size + 1), cum_var_exp, where='mid', label = 'Cumulative explained variance')
plt.ylabel('Explained Variance Ratio')
plt.xlabel('Principal Components')
plt.legend(loc = 'best')
plt.tight_layout()
plt.show()
```



4.11 Capturing 90% variance of the data

```
# Capturing 90% variance of the data
pca = PCA(n_components = 0.90)
X_train_reduced = pca.fit_transform(X_train)
X_test_reduced = pca.transform(X_test)
```

```
print(X_train_reduced.shape)
print(X_test_reduced.shape)
```

```
(335, 20)
(84, 20)
```

5. DESIGN TRAIN AND TEST BASIC MACHINE LEARNING CLASSIFIERS

5.1 Train and test model

```
def model(mod, method, X_train, X_test, y_train, y_test, of_type, index, scale, report, save_model):

    if report == "yes":
        print (mod)
        print ("********************************************************************************")

    if method == 'CatBoost' or method == 'LGBM':

      mod.fit(X_train, y_train) # Fit the model on Training set
    else:
      mod.fit(X_train, y_train) # Fit the model on Training set

    from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, recall_score, precision_scor

    if of_type == "coef":
        # Intercept and Coefficients
        print("The intercept for our model is {}".format(model.intercept_), "\n")

        for idx, col_name in enumerate(X_train.columns):
            print("The coefficient for {} is {}".format(col_name, model.coef_.ravel()[idx]))
```

```
    y_pred = mod.predict(X_test) # Predict on Test set

    # Initialise mc_logloss
    mc_logloss = 1.00
    if method != 'Ridge':
      y_predictions = mod.predict_proba(X_test)

    train_accuracy_score = mod.score(X_train, y_train)
    test_accuracy_score = mod.score(X_test, y_test)

    precision_score = precision_score(y_test, y_pred, average='weighted')
    recall_score = recall_score(y_test, y_pred, average='weighted')
    f1_score = f1_score(y_test, y_pred, average='weighted')

    if method != 'Ridge':
      mc_logloss = logloss(y_test, y_predictions, eps=1e-15)

    if report == "yes":
      # Model - Confusion matrix
      model_cm = confusion_matrix(y_test, y_pred)
```

5.2 Train and test all models

```
import lightgbm as lgb

def models(X_train_common, X_test_common, y_train, y_test, scale):

    # define classification models
    mods=[['LogisticReg',LogisticRegression(solver='lbfgs', multi_class='multinomial', random_state = 1)
        ['KNN',KNeighborsClassifier(n_neighbors = 3)],
        ['SVC',SVC(kernel = 'rbf', probability=True)],
        ['RandomForest',RandomForestClassifier(n_estimators=10, random_state=1)],
        ['AdaBoost',AdaBoostClassifier(n_estimators=100, learning_rate=0.25, random_state=1)],
        ['GradientBoosting',GradientBoostingClassifier(loss='deviance', n_estimators=50, learning_rate=0
                                    random_state=1)]
    ]

    ResDF = pd.DataFrame()
    i = 1
    for name, classifier in mods:
        # Train and Test the model
        itr_resDF = model(classifier, name, X_train_common, X_test_common, y_train, y_test, 'none', i, s
```

```
        # Store the accuracy results for each model in a dataframe for final comparison
        ResDF = pd.concat([ResDF, itr_resDF])
        i = i+1

    return ResDF
```

5.3 Model with Hyperparameter Tuning

```
def tuning_model(name, mod, X_train, y_train, param_grid):

    start = time.time()  # note the start time

    # Before starting with grid search we need to create a scoring function. This is accomplished using
    mll_scorer = metrics.make_scorer(logloss, greater_is_better=False, needs_proba=True)

    # define grid search
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    if name == 'LGBM':
      grid_search = RandomizedSearchCV(estimator=mod, param_distributions=param_grid, n_iter=100, n_jobs
                                       scoring = mll_scorer, error_score=0)
    else:
      grid_search = GridSearchCV(estimator=mod, param_grid=param_grid, n_jobs=-1, cv=cv,
                                 scoring = mll_scorer, error_score=0)

    model_grid_result = grid_search.fit(X_train, y_train)

    # summarize results
    print("Best F1_Score: %f using %s" % (model_grid_result.best_score_, model_grid_result.best_params_)
```

```
    means = model_grid_result.cv_results_['mean_test_score']
    stds = model_grid_result.cv_results_['std_test_score']
    params = model_grid_result.cv_results_['params']
    for mean, stdev, param in zip(means, stds, params):
      if param == model_grid_result.best_params_:
        print("%f (%f) with: %r" % (mean, stdev, param))
        print("95% Confidence interval range: ({0:.4f} %, {1:.4f} %)".format(mean-(2*stdev), mean+(2*std

    end = time.time()  # note the end time
    duration = end - start  # calculate the total duration
    print("Total duration" , duration, "\n")

    return model_grid_result.best_estimator_
```

5.4 Logistic Regression Model

	precision	recall	f1-score	support
0	0.62	0.56	0.59	9
1	0.38	0.42	0.40	19

```
    macro avg       0.39      0.39      0.39        84
 weighted avg       0.48      0.49      0.49        84
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

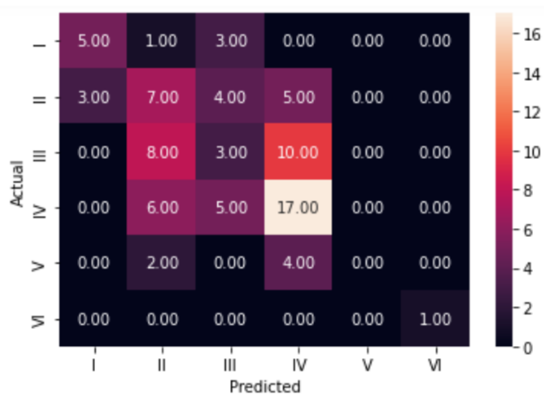
Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-Score	Multi-Class Logloss
Logistic Reg no Samp	0.689552	0.488095	0.483838	0.488095	0.485398	1.244126

5.5 Building a Random Forest Classifier on Training set

```
rf = RandomForestClassifier(n_estimators=10, random_state=1)

# Train and Test the model
rf_df = model(rf, 'Random Forest org', X_train, X_test, y_train, y_test, 'none', 2, 'no', 'yes', 'no')

#Store the accuracy results for each model in a dataframe for final comparison
metricsDF = pd.concat([metricsDF,rf_df])
metricsDF
```

```
RandomForestClassifier(n_estimators=10, random_state=1)
**************************************************************************
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```




```
              precision    recall  f1-score   support

           0       0.62      0.56      0.59         9
           1       0.29      0.37      0.33        19
```

```
weighted avg       [illegible]
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

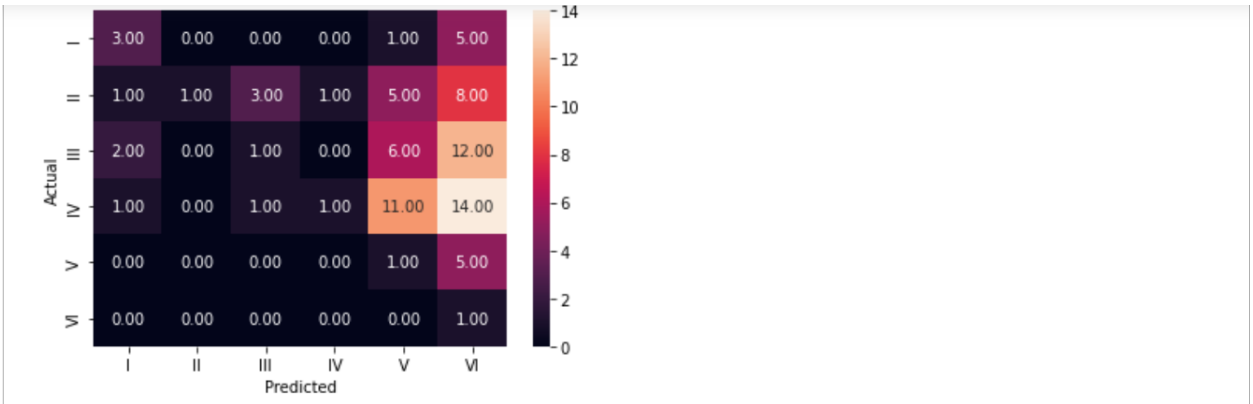
	Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-Score	Multi-Class Logloss
1	Logistic Reg no Samp	0.689552	0.488095	0.483838	0.488095	0.485398	1.244126
2	Random Forest org	0.985075	0.392857	0.352249	0.392857	0.367323	5.495064

5.6 Building a Logistic Regression model with Sampling

```
lr = LogisticRegression(solver='lbfgs', multi_class='multinomial', random_state = 1)

# Train and Test the model
lr_df = model(lr, 'Logistic Reg Samp', X_train_up, X_test, y_train_up, y_test, 'none', 3, 'no', 'yes', '

#Store the accuracy results for each model in a dataframe for final comparison
metricsDF = pd.concat([metricsDF,lr_df])
metricsDF
```

```
LogisticRegression(multi_class='multinomial', random_state=1)
***************************************************************************
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lb
fgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```



	precision	recall	f1-score	support
0	0.43	0.33	0.38	9
1	1.00	0.05	0.10	19
2	0.20	0.05	0.08	21
3	0.50	0.04	0.07	28
4	0.04	0.17	0.07	6
5	0.02	1.00	0.04	1
accuracy			0.10	84
macro avg	0.37	0.27	0.12	84
weighted avg	0.49	0.10	0.11	84

	Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-Score	Multi-Class Logloss
1	Logistic Reg no Samp	0.689552	0.488095	0.483838	0.488095	0.485398	1.244126
2	Random Forest org	0.985075	0.392857	0.352249	0.392857	0.367323	5.495064
3	Logistic Reg Samp	0.563422	0.095238	0.492016	0.095238	0.109530	3.054866

5.7 Training and testing models with original data

```
models(X_train, X_test, y_train, y_test, 'no')
```

	Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-Score	Multi-Class Logloss
1	LogisticReg	0.689552	0.488095	0.483838	0.488095	0.485398	1.244126
2	KNN	0.695522	0.404762	0.408230	0.404762	0.393287	9.991412
3	SVC	0.674627	0.500000	0.524860	0.500000	0.488264	1.221855
4	RandomForest	0.985075	0.392857	0.352249	0.392857	0.367323	5.495064
5	AdaBoost	0.426866	0.345238	0.361391	0.345238	0.347262	1.362507
6	GradientBoosting	0.949254	0.440476	0.442842	0.440476	0.439388	1.347754

5.8 Training and Testing all models with oversampling data

```
models(X_train_up, X_test, y_train_up, y_test, 'no')
```

	Method	Train Accuracy	Test Accuracy	Precision	Recall	F1-Score	Multi-Class Logloss
1	LogisticReg	0.563422	0.095238	0.492016	0.095238	0.109530	3.054866
2	KNN	0.862832	0.142857	0.028261	0.142857	0.045133	26.424715
3	SVC	0.328909	0.250000	0.062500	0.250000	0.100000	1.780355
4	RandomForest	0.998525	0.404762	0.384650	0.404762	0.384403	5.491840
5	AdaBoost	0.368732	0.452381	0.308730	0.452381	0.363366	1.323568
6	GradientBoosting	0.969027	0.464286	0.563555	0.464286	0.437437	1.466504

5.9 Defining regressor models

```
mod=[['LogisticReg',LogisticRegression()],
    ['KNN',KNeighborsClassifier()],
    ['SVC',SVC()],
    ['RandomForest',RandomForestClassifier()],
    ['AdaBoost',AdaBoostClassifier()],
    ['GradientBoosting',GradientBoostingClassifier()],
]

# define model parameters

lr_param_grid = {'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
                 'penalty': ['l2'],
                 #'penalty': ['none', 'l1', 'l2', 'elasticnet'],
                 'C': [100, 10, 1.0, 0.1, 0.01]}
                 #'class_weight':['none','balanced'],
                 #'multi_class':['ovr', 'multinomial']}
knn_param_grid = {'n_neighbors': range(3, 21, 2),
                  'weights': ['uniform', 'distance'],
                  'metric': ['euclidean', 'manhattan', 'minkowski']}
svc_param_grid = {'kernel': ['poly', 'rbf', 'sigmoid'],
                  'C': [50, 10, 1.0, 0.1, 0.01],
                  'gamma': ['scale'],
                  'decision_function_shape': ['ovo', 'ovr']}
rf_param_grid = {'n_estimators': [10, 100, 1000],
                 'max_features': ['auto', 'sqrt', 'log2']}
                 #'class_weight':['balanced','balanced_subsample','none']}
adb_param_grid = {'n_estimators': np.arange(30,100,10),
                  'learning_rate': np.arange(0.1,1,0.5)}
gb_param_grid = {'n_estimators': [10, 50, 100, 500],
                 'learning_rate': [0.0001, 0.001, 0.01, 0.1, 1.0],
                 'subsample':[0.5, 0.7, 1.0],
                 'max_depth': [3, 7, 9]}


for name, classifier in mod:
    if name == 'LogisticReg':
        lr_best_estimator = tuning_model(name, classifier, X_train, y_train, lr_param_grid)
    elif name == 'KNN':
        knn_best_estimator = tuning_model(name, classifier, X_train, y_train, knn_param_grid)
    elif name == 'SVC':
```

```
Best F1_Score: -1.083363 using {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
-1.083363 (0.370224) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
95% Confidence interval range: (-1.8238 %, -0.3429 %)
Total duration 62.32723903656006
```

```
Best F1_Score: -1.701869 using {'metric': 'euclidean', 'n_neighbors': 19, 'weights': 'distance'}
-1.701869 (1.016063) with: {'metric': 'euclidean', 'n_neighbors': 19, 'weights': 'distance'}
95% Confidence interval range: (-3.7340 %, 0.3303 %)
Total duration 10.940994501113892
```

```
Best F1_Score: 0.000000 using {'C': 50, 'decision_function_shape': 'ovo', 'gamma': 'scale', 'kerne
l': 'poly'}
0.000000 (0.000000) with: {'C': 50, 'decision_function_shape': 'ovo', 'gamma': 'scale', 'kernel': 'p
oly'}
95% Confidence interval range: (0.0000 %, 0.0000 %)
Total duration 16.55791997909546
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_split.py:676: UserWarning: The least
populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_split.py:676: UserWarning: The least
populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_split.py:676: UserWarning: The least
populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(
```

```
Best F1_Score: -1.048966 using {'max_features': 'auto', 'n_estimators': 1000}
-1.048966 (0.361343) with: {'max_features': 'auto', 'n_estimators': 1000}
95% Confidence interval range: (-1.7717 %, -0.3263 %)
Total duration 175.42208456993103
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_split.py:676: UserWarning: The least
populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_split.py:676: UserWarning: The least
populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_split.py:676: UserWarning: The least
populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(
```

```
Best F1_Score: -1.210071 using {'learning_rate': 0.1, 'n_estimators': 30}
-1.210071 (0.407122) with: {'learning_rate': 0.1, 'n_estimators': 30}
95% Confidence interval range: (-2.0243 %, -0.3958 %)
Total duration 49.80737614631653
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_split.py:676: UserWarning: The least
populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_split.py:676: UserWarning: The least
populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_split.py:676: UserWarning: The least
```

Observations:

- We can see that gradient descent boost is performing better for sampling.
- In the report regarding the logic behind using upsampling like -In order to improve our model validation, we tried replicating the data for and applied upsampling technique as there was only one value for severity level 6 of Potential Accident Level which when splitted into train and test would get included in either of the two so to avoid this bias, the records were balanced using upsampling technique and then models were applied and we observe that Gradient Boost performs far better than any other models applied so far

Steps for next Milestone:

We will be using Bi-LSTM for next milestones.

Building Chatbot

We will be exploring Tkinter – python GUI programming tool to build our chatbot for this project. We will explore how we can deploy a model and check real-time predictions using Tkinter. We will first build a classification model that will classify on Potential Accident severity level. Then we will make a GUI using Tkinter and will check predictions on new data points. Methodology-

[1]. Creating GUI

Tkinter is a library written in Python that is widely used to create GUI applications. It is very easy to build GUI using Tkinter and the process is even faster. Tkinter has several widgets that can be used while developing GUI. These include buttons, radio buttons, checkboxes, etc. We will see how we can make a GUI Tkinter after we build the model.

[2]. Model Building

As discussed earlier, we will be proceeding ahead with Bidirectional LSTM model as the accuracy is higher than compared to other models. Once we are done with the model validation, we pickle this model that would be used to compute predictions for new data points.

[3]. Computing Predictions Using the Tkinter GUI in Real-Time

To predict the class, we will need to provide input in the same way as we did while training. So we will create some functions that will perform text preprocessing and then predict the class. After predicting the class, we will get a random response from the list of intents. Now we will develop a graphical user interface. We will take the input message from the user and then use the helper functions we have created to get the response from the bot and display it on the GUI.

Milestone – 2

Data Modelling

While building a predictive model we follow several different steps. We first do exploratory data analysis to understand the data well and do the required preprocessing. After the data gets ready we do modelling and develop a predictive model. This model is then used to compute prediction on the testing data and the results are evaluated using different error metrics. In this project, for modelling part we have proceeded with different ML models, Neural Network and Bidirectional LSTM.

ML Models and Neural Network:

Applying different ML models and Neural Network will give us a comparative overview on the analysis.To start off we proceed ahead with further pre-processing and then built the respective model

Methodology

Pre-processing-

[1.] Date column being splitted by Date, Month and Year

[2.] Description column had to undergo few data treatment like pre-processing (lowercase, stopwords, stemming, tensors) and word embedding using glove.

[3.]Created dummies for each column except Description and imputed values for each unique level of Potential Accident Level.

6.1 Importing Libraries , Dataset and printing Dataframe

```
#importing necessary library
import pandas as pd
import numpy as np

# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
```

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/conten
t/drive/", force_remount=True).
```

```
# Let's import the Data set
df=pd.read_csv('/content/drive/MyDrive/CAPSTONE/Data Set - industrial_safety_and_health_database_with_ac
```

```
# printing the dataframe file.
df
```

	Unnamed: 0	Data	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Genre	Employee or Third Party	Critical Risk	Description
0	0	2016-01-01 00:00:00	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
1	1	2016-01-02 00:00:00	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
2	2	2016-01-06 00:00:00	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...
3	3	2016-01-08 00:00:00	Country_01	Local_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...

6.2 **Dividing Date column in Day, Month and Year**

```
year =[]
month =[]
date=[]
for x in range(df.shape[0]):
    h = df['Data'][x].split()
    k = h[0].split('-')
    year.append(int(k[0]))
    month.append(int(k[1]))
    date.append(int(k[2]))
```

```
dates = list(zip(year,month,date))
df_date = pd.DataFrame(dates, columns = ['Year','Month','Date'])
df_date
```

	Year	Month	Date
0	2016	1	1
1	2016	1	2
2	2016	1	6
3	2016	1	8
4	2016	1	10
...	...	...	...
420	2017	7	4
421	2017	7	4
422	2017	7	5
423	2017	7	6
424	2017	7	9

425 rows × 3 columns

6.3 Concatenating the date, Month and Year to new_df

```
df_new =pd.concat([df,df_date],axis=1)
df_new
```

	Unnamed: 0	Data	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Genre	Employee or Third Party	Critical Risk	Description	Ye
0	0	2016-01-01 00:00:00	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...	20
1	1	2016-01-02 00:00:00	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	20
2	2	2016-01-06 00:00:00	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	20
											Being 9:45	

```
col = df_new.columns.tolist()
col
```

```
['Unnamed: 0',
 'Data',
 'Countries',
 'Local',
 'Industry Sector',
 'Accident Level',
 'Potential Accident Level',
 'Genre',
 'Employee or Third Party',
 'Critical Risk',
 'Description',
 'Year',
 'Month',
 'Date']
```

6.4 Concatenating the columns in one dataframe

```
col1 =col[:2]+col[-3:]+col[2:-3]
col1
```

```
['Unnamed: 0',
 'Data',
 'Year',
 'Month',
 'Date',
 'Countries',
 'Local',
 'Industry Sector',
 'Accident Level',
 'Potential Accident Level',
 'Genre',
 'Employee or Third Party',
 'Critical Risk',
 'Description']
```

```
df_latest = df_new[col1]
```

```
df_latest
```

	Unnamed: 0	Data	Year	Month	Date	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Genre	Employee or Third Party	Critica Risl
0	0	2016-01-01 00:00:00	2016	1	1	Country_01	Local_01	Mining	I	IV	Male	Third Party	Presse
1	1	2016-01-02 00:00:00	2016	1	2	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurize System
2	2	2016-01-06 00:00:00	2016	1	6	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manua Tool

6.5 Dropping Unnamed: 0, and Data column from dataframe

```
df_latest.drop(['Unnamed: 0','Data'], axis=1,inplace = True)
df_latest.head(3)
```

	Year	Month	Date	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Genre	Employee or Third Party	Critical Risk	Description
0	2016	1	1	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
1	2016	1	2	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
2	2016	1	6	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...

6.6 Value counts for column 'Local', ‘Countries’

```
df_latest['Local'].value_counts()
```

```
Local_03    90
Local_05    59
Local_01    57
Local_04    56
Local_06    46
Local_10    44
Local_08    27
Local_02    24
Local_07    14
Local_12     4
Local_09     2
Local_11     2
Name: Local, dtype: int64
```

```
df_latest['Countries'].value_counts()
```

```
Country_01    251
Country_02    130
Country_03     44
Name: Countries, dtype: int64
```

6.7 Mapping potential Accident level into the dataframe

Opting for Potential Accidents Level as our target variable as Accident Level didn't have sufficient data points for each severity level which was evident by most of the data falling under severity 1 contributing to biasness. Concluding this basis on the lowest accuracy, precision and recall when model was applied on raw data as well as over sampled data considering accident level as target variable. Compare to Accident Level, Potential Accident Level has somewhat balanced values except for the fact Level 6 has only 1 record. Hence, dropped Accident Level.

```
df_latest['Potential_Accident_Level'] = df_latest['Potential Accident Level'].map(Potential_Accident_Lev
```

```
df_latest['Potential_Accident_Level'].value_counts()
```

```
4    143
3    106
2     95
1     49
5     31
6      1
Name: Potential_Accident_Level, dtype: int64
```

6.8 Apply get_dummies to columns to get data in the form of 0's and 1's

```
df_latest = pd.get_dummies(df_latest, columns=['Date','Year','Countries','Critical Risk','Local','Indust
df_latest = pd.get_dummies(df_latest, columns=['Month'],prefix='Month',drop_first=True)
```

```
df_latest.drop(['Accident Level','Potential Accident Level'],axis=1,inplace=True)
```

```
df_latest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 425 entries, 0 to 424
Data columns (total 94 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Description                             425 non-null    object
 1   Potential_Accident_Level                425 non-null    int64
 2   Dummy_2                                 425 non-null    uint8
 3   Dummy_3                                 425 non-null    uint8
 4   Dummy_4                                 425 non-null    uint8
 5   Dummy_5                                 425 non-null    uint8
```

7.1 Cleaning up the data, removing stopwords, remove punctuation, converting them into tokens

```
import string
import re
import os
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.tokenize import TweetTokenizer
from nltk.corpus import stopwords, twitter_samples

tweet_tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True, reduce_len=True)

stopwords_english = stopwords.words('english')


def process_tweet(tweet):
    '''
    Input:
        tweet: a string containing a tweet
    Output:
        tweets_clean: a list of words containing the processed tweet
```

```
# remove stock market tickers like $GE
tweet = re.sub(r'\$\w*', '', tweet)
#remove numbers
tweet = re.sub(r'\d+', '', tweet)
# remove old style retweet text "RT"
tweet = re.sub(r'^RT[\s]+', '', tweet)
# remove hyperlinks
tweet = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet)
# remove hashtags
# only removing the hash # sign from the word
tweet = re.sub(r'#', '', tweet)
# tokenize tweets
tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True, reduce_len=True)
tweet_tokens = tokenizer.tokenize(tweet)
### START CODE HERE ###
tweets_clean = []
for word in tweet_tokens:
    if (word not in stopwords_english and # remove stopwords
```

7.2 Describing

```
df_latest['Description'][0]
```

```
'While removing the drill rod of the Jumbo 08 for maintenance, the supervisor proceeds to loosen the s
upport of the intermediate centralizer to facilitate the removal, seeing this the mechanic supports on
e end on the drill of the equipment to pull with both hands the bar and accelerate the removal from th
is, at this moment the bar slides from its point of support and tightens the fingers of the mechanic b
etween the drilling bar and the beam of the jumbo.'
```

```
b = process_tweet(df_latest['Description'][0])
b
```

```
['removing',
 'drill',
 'rod',
 'jumbo',
 'maintenance',
 'supervisor',
 'proceeds',
 'loosen',
```

7.3 Check length of all words in description

```
all_word = []

for row in df_latest['Description']:
    for word in process_tweet(row):
        all_word.append(word)
```

```
len(all_word)
```

```
14120
```

```
import os
path = os.getcwd()
path
```

```
'/content'
```

7.4 Using GloVe Embedding of 50 dimension

```
import os
import gensim
from gensim.test.utils import datapath, get_tmpfile
from gensim.models import KeyedVectors # Gensim contains word2vec models and processing tools

path = os.getcwd()

glove_file = '/content/drive/MyDrive/glove.6B.50d.txt' # This is a GloVe model


tmp_file = '/content/drive/MyDrive/word2vec.glove.6B.50d.txt'


from gensim.scripts.glove2word2vec import glove2word2vec
_ =  glove2word2vec(glove_file, tmp_file)  # Converting the GloVe file into a Word2Vec file
model = KeyedVectors.load_word2vec_format(tmp_file)
```

7.5 Checking what the embedding looks like

```
wordEmbed = model['cat']
print(wordEmbed.shape)
print(wordEmbed)
```

```
(50,)
[ 0.45281  -0.50108  -0.53714  -0.015697  0.22191   0.54602  -0.67301
 -0.6891    0.63493  -0.19726   0.33685   0.7735    0.90094   0.38488
  0.38367   0.2657   -0.08057   0.61089  -1.2894   -0.22313  -0.61578
  0.21697   0.35614   0.44499   0.60885  -1.1633   -1.1579    0.36118
  0.10466  -0.78325   1.4352    0.18629  -0.26112   0.83275  -0.23123
  0.32481   0.14485  -0.44552   0.33497  -0.95946  -0.097479  0.48138
 -0.43352   0.69455   0.91043  -0.28173   0.41637  -1.2609    0.71278
  0.23782 ]
```

7.6 Converting Description into GloVe embeddings.

```
data_list = list()
for row in df_latest['Description']:
    sentence = np.zeros(50)
    count = 0
    for word in process_tweet(row):
        try:
            sentence += model[word]
            count += 1
        except KeyError:
            continue
    data_list.append(sentence / count)
```

7.7 Splitting the data into training and test set of ratio 80:20

```
from sklearn.model_selection import train_test_split
X_tr, X_te, y_tr, y_te = train_test_split(np.array(data_list), y, test_size=0.20,random_state=42)
```

```
print(f"x_train.shape:{X_tr.shape} ,y_train.shape:{y_tr.shape}")
print(f"x_test.shape:{X_te.shape},y_test.shape:{y_te.shape}")
```

```
x_train.shape:(340, 50) ,y_train.shape:(340,)
x_test.shape:(85, 50),y_test.shape:(85,)
```

7.8 Method to store Accuracy of Model.

```
acc = []
mod = []
def get_accuracy(y,x):
    global t
    acc.append(x)
    mod.append(y)
    temp=pd.DataFrame(mod,columns=['Model'])
    temp1 = pd.DataFrame(acc,columns=['accuracy'])
    t =temp.join(temp1)
    print(t)
```

7.9 Applying RandomForestRegressor Model

```
from sklearn.ensemble import  RandomForestClassifier
rf =  RandomForestClassifier()
```

```
rf.fit(X_tr,y_tr)
```

```
RandomForestClassifier()
```

```
print(rf.score(X_tr,y_tr))
print(rf.score(X_te,y_te))
```

```
0.9970588235294118
0.4
```

```
get_accuracy('RandomForestClassifier',rf.score(X_te,y_te))
```

```
                    Model  accuracy
0  RandomForestClassifier       0.4
```

7.10 Taking GloVe embeddings

```
b = np.array(data_list)
```

```
df = pd.DataFrame(b)
```

```
df
```

	0	1	2	3	4	5	6	7	8	9	...	40	
0	0.046364	-0.029248	0.222147	-0.069934	-0.007058	0.178736	-0.022133	0.027935	0.297329	-0.117455	...	-0.086675	0.(
1	0.517001	0.210823	0.313585	-0.238873	-0.080969	0.442216	0.183545	0.063399	0.154941	0.317588	...	0.252190	0.(
2	0.053373	-0.015110	0.161758	-0.138729	0.164716	0.183081	-0.223473	0.132254	0.235123	-0.083697	...	-0.077956	0.(
3	0.138974	-0.054045	0.219032	-0.018591	0.175420	0.094466	0.073502	0.116270	0.012112	-0.280907	...	0.000397	0.(
4	0.040683	0.194368	0.381006	-0.076749	0.186348	0.380040	-0.011421	-0.056856	0.153616	-0.198066	...	-0.076633	0.(
...	...	...	...	...	...	...	...	...	...	...	...	...	
420	0.021859	0.037871	0.279773	-0.158140	0.440398	0.294467	-0.023263	-0.064917	0.028563	-0.447277	...	0.066633	0.(
421	0.127191	-0.025613	0.172690	-0.056459	-0.029684	0.139266	-0.239016	0.012693	0.023200	-0.065939	...	-0.154663	0.:
422	0.400082	-0.100790	0.015473	-0.114683	0.096129	0.074122	-0.242359	0.203369	0.147597	-0.275508	...	-0.282320	0.(
423	-0.055878	0.012568	0.219108	-0.102659	0.127429	0.412077	0.018652	0.143614	0.051245	0.089297	...	-0.037024	0.:
424	0.145441	0.052734	0.167359	-0.137173	0.309313	-0.192620	-0.370408	0.227082	0.053699	-0.440744	...	-0.063406	0.(

425 rows × 50 columns

```
new_df = pd.concat([df_latest,df],axis=1)
```

```
new_df.drop(['Description'],axis=1,inplace=True)
```

```
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 425 entries, 0 to 424
Columns: 143 entries, Potential_Accident_Level to 49
dtypes: float64(50), int64(1), uint8(92)
memory usage: 207.6 KB
```

```
new_df.shape
```

```
(425, 143)
```

```
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 425 entries, 0 to 424
Columns: 143 entries, Potential_Accident_Level to 49
dtypes: float64(50), int64(1), uint8(92)
memory usage: 207.6 KB
```

```
df_final = new_df.copy()
```

```
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 425 entries, 0 to 424
Columns: 143 entries, Potential_Accident_Level to 49
dtypes: float64(50), int64(1), uint8(92)
memory usage: 207.6 KB
```

```
y.value_counts()
```

```
4    143
3    106
2     95
1     49
5     31
6      1
Name: Potential_Accident_Level, dtype: int64
```

```
y_list = list(y)
d = y_list.index(6)
d
```

```
307
```

```
X
```

	Potential_Accident_Level	Dummy_2	Dummy_3	Dummy_4	Dummy_5	Dummy_6	Dummy_7	Dummy_8	Dummy_9	Dummy
0	4	0	0	0	0	0	0	0	0	
1	4	1	0	0	0	0	0	0	0	
2	3	0	0	0	0	1	0	0	0	
3	1	0	0	0	0	0	0	1	0	
4	4	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	
420	3	0	0	1	0	0	0	0	0	
421	2	0	0	1	0	0	0	0	0	
422	2	0	0	0	1	0	0	0	0	
423	2	0	0	0	0	1	0	0	0	

7.11 Apply RandomForestRegressor Model using GloVe

```
rf1 =  RandomForestClassifier()
```

```
rf1.fit(X_tr,y_tr)
```

```
RandomForestClassifier()
```

```
rf1.score(X_tr,y_tr)
```

```
0.9966329966329966
```

```
rf1.score(X_te,y_te)
```

```
0.484375
```

```
get_accuracy('RandomForestClassifier GloVe',rf1.score(X_te,y_te))
```

```
                          Model  accuracy
  0       RandomForestClassifier  0.400000
```

```
   1  RandomForestClassifier GloVe  0.484375
```

```
pred = rf1.predict(X_te)
pred = pred.astype(int)
pred
```

```
array([2, 4, 4, 2, 1, 1, 3, 3, 4, 3, 4, 4, 3, 4, 2, 4, 4, 4, 4, 2, 2, 4,
       4, 2, 4, 3, 3, 2, 2, 1, 3, 4, 4, 1, 4, 2, 4, 4, 3, 2, 4, 2, 1, 4,
       2, 2, 4, 1, 1, 4, 4, 4, 4, 5, 3, 3, 4, 1, 4, 3, 4, 2, 2, 4, 4, 4,
       3, 4, 2, 4, 4, 4, 3, 2, 1, 4, 4, 3, 1, 2, 2, 4, 4, 4, 2, 3, 4, 4,
       4, 1, 4, 3, 2, 3, 4, 4, 4, 3, 4, 4, 4, 4, 3, 4, 1, 3, 4, 2, 3, 4,
       1, 4, 1, 3, 3, 3, 4, 4, 4, 4, 2, 4, 4, 3, 2, 4, 3, 4])
```

```
from sklearn.metrics import classification_report

print(classification_report(y_te, pred))
```

```
                precision    recall  f1-score   support

             1       0.86      0.71      0.77        17
             2       0.42      0.36      0.38        28
             3       0.50      0.37      0.43        35
             4       0.41      0.70      0.52        37
             5       1.00      0.09      0.17        11

      accuracy                           0.48       128
     macro avg       0.64      0.45      0.45       128
  weighted avg       0.55      0.48      0.47       128
```

7.12 Apply DecisionTreeClassifier Model

```
from sklearn.tree import DecisionTreeClassifier

dtree=DecisionTreeClassifier()
```

```
dtree.fit(X_tr,y_tr)
```

```
DecisionTreeClassifier()
```

```
dtree.score(X_tr,y_tr)
```

```
0.9966329966329966
```

```
dtree.score(X_te,y_te)
```

```
0.3828125
```

```
get_accuracy('DecisionTreeClassifier',dtree.score(X_te,y_te))
```

```
                          Model  accuracy
0       RandomForestClassifier  0.400000
1  RandomForestClassifier GloVe  0.484375
2       DecisionTreeClassifier  0.382812
```

```
pred = dtree.predict(X_te)
pred = pred.astype(int)
pred
```

```
array([3, 1, 3, 3, 1, 1, 4, 4, 1, 5, 2, 2, 3, 1, 2, 1, 3, 4, 3, 1, 3, 5,
       4, 4, 2, 4, 4, 3, 3, 1, 3, 4, 3, 4, 3, 1, 4, 3, 1, 5, 3, 3, 3, 4,
       5, 2, 5, 4, 1, 4, 4, 4, 3, 5, 3, 4, 4, 1, 2, 3, 4, 3, 2, 3, 4, 2,
       4, 3, 4, 4, 4, 3, 2, 4, 1, 4, 2, 1, 1, 4, 4, 2, 3, 4, 3, 2, 4, 4,
       4, 4, 1, 2, 4, 2, 3, 4, 2, 2, 2, 4, 3, 3, 4, 2, 1, 2, 1, 2, 1, 3,
       1, 3, 1, 4, 1, 4, 4, 3, 5, 4, 4, 4, 4, 4, 4, 4, 3, 3])
```

```
from sklearn.metrics import classification_report

print(classification_report(y_te, pred))
```

```
              precision    recall  f1-score   support

           1       0.45      0.59      0.51        17
           2       0.30      0.21      0.25        28
           3       0.36      0.34      0.35        35
           4       0.43      0.54      0.48        37
           5       0.14      0.09      0.11        11

    accuracy                           0.38       128
   macro avg       0.34      0.36      0.34       128
weighted avg       0.36      0.38      0.37       128
```

7.13 Apply BernoulliNB Model

```
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score
clf = BernoulliNB()
clf.fit(X_tr, y_tr)
pred = clf.predict(X_te)
print(accuracy_score(y_te, pred))
```

```
0.4140625
```

```
clf.score(X_te,y_te)
```

```
0.4140625
```

```
get_accuracy('BernoulliNB',clf.score(X_te,y_te))
```

```
                          Model  accuracy
0       RandomForestClassifier  0.400000
1  RandomForestClassifier GloVe  0.484375
2       DecisionTreeClassifier  0.382812
3                   BernoulliNB  0.414062
```

7.14 Apply BaggingClassifier Model

```
from sklearn.ensemble import BaggingClassifier
bgcl=BaggingClassifier()
```

```
bgcl.fit(X_tr,y_tr)
```

```
BaggingClassifier()
```

```
bgcl.score(X_tr,y_tr)
```

```
0.9831649831649831
```

```
bgcl.score(X_te,y_te)
```

```
0.375
```

```
get_accuracy('BaggingClassifier',bgcl.score(X_te,y_te))
```

```
                          Model  accuracy
0        RandomForestClassifier  0.400000
0        RandomForestClassifier  0.400000
1  RandomForestClassifier GloVe  0.484375
2        DecisionTreeClassifier  0.382812
3                  BernoulliNB  0.414062
4             BaggingClassifier  0.375000
```

7.15 Data Balance

```
y.value_counts()
```

```
4    143
3    106
2     95
1     49
5     31
6      1
Name: Potential_Accident_Level, dtype: int64
```

```
X.head(5)
```

	Dummy_2	Dummy_3	Dummy_4	Dummy_5	Dummy_6	Dummy_7	Dummy_8	Dummy_9	Dummy_10	Dummy_11	...	
0	0	0	0	0	0	0	0	0	0	0	...	-0.0866
1	1	0	0	0	0	0	0	0	0	0	...	0.2521
2	0	0	0	0	1	0	0	0	0	0	...	-0.0779
3	0	0	0	0	0	0	1	0	0	0	...	0.0003

```
K = pd.concat([X,y],axis=1)
P=K[K['Potential_Accident_Level'] == 6]
P
```

	Dummy_2	Dummy_3	Dummy_4	Dummy_5	Dummy_6	Dummy_7	Dummy_8	Dummy_9	Dummy_10	Dummy_11	...	
307	0	0	0	0	0	0	0	0	0	0	...	-0.05

1 rows × 143 columns

```
J = K.append([P]*10,ignore_index=True)
J
```

	Dummy_2	Dummy_3	Dummy_4	Dummy_5	Dummy_6	Dummy_7	Dummy_8	Dummy_9	Dummy_10	Dummy_11	...	
0	0	0	0	0	0	0	0	0	0	0	...	0.00
1	1	0	0	0	0	0	0	0	0	0	...	0.05
2	0	0	0	0	1	0	0	0	0	0	...	0.05

```
J['Potential_Accident_Level'].value_counts()
```

```
4    143
3    106
2     95
1     49
5     31
6     11
Name: Potential_Accident_Level, dtype: int64
```

```
J.drop('Potential_Accident_Level',axis=1,inplace=True)
```

```
X=J
```

```
X.shape
```

```
(435, 142)
```

```
len(Y)
```

```
435
```

```
y =np.array(Y)
```

```
X.columns
```

```
Index([ 'Dummy_2',  'Dummy_3',  'Dummy_4',  'Dummy_5',  'Dummy_6',  'Dummy_7',
        'Dummy_8',  'Dummy_9', 'Dummy_10', 'Dummy_11',
       ...
               40,         41,         42,         43,         44,         45,
               46,         47,         48,         49],
      dtype='object', length=142)
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X,y, test_size = 0.20, random_state = 55)
```

7.16 Applying SMOTE for OverSampling of data.

```
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
x_train,y_train = oversample.fit_resample(X_train.values, Y_train.ravel())
```

```
target=pd.DataFrame(y_train)
target[0].value_counts()
```

```
2    119
1    119
4    119
5    119
3    119
6    119
Name: 0, dtype: int64
```

7.17 Apply RandomForestClassifier using SMOTE

```
rf2 =  RandomForestClassifier()
rf2.fit(x_train,y_train)
rf2.score(X_test,Y_test)
```

```
0.41379310344827586
```

```
get_accuracy('RandomForestClassifier SMOTE',rf2.score(X_test,Y_test))
```

```
                         Model  accuracy
0       RandomForestClassifier  0.400000
1  RandomForestClassifier GloVe  0.484375
2       DecisionTreeClassifier  0.382812
3                  BernoulliNB  0.414062
4            BaggingClassifier  0.375000
5  RandomForestClassifier SMOTE  0.413793
```

7.18 Apply DecisionTreeClassifier using SMOTE

```
from sklearn.tree import DecisionTreeClassifier

dtree1=DecisionTreeClassifier()
dtree1.fit(x_train,y_train)
dtree1.score(X_test,Y_test)
```

```
0.3218390804597701
```

```
get_accuracy('DecisionTreeClassifier SMOTE',dtree1.score(X_test,Y_test))
```

```
                         Model  accuracy
0       RandomForestClassifier  0.400000
1  RandomForestClassifier GloVe  0.484375
2       DecisionTreeClassifier  0.382812
3                  BernoulliNB  0.414062
4            BaggingClassifier  0.375000
5  RandomForestClassifier SMOTE  0.413793
6  DecisionTreeClassifier SMOTE  0.321839
```

7.19 Apply BernoulliNB using SMOTE

```
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score
clf = BernoulliNB()
clf.fit(x_train,y_train)
pred = clf.predict(X_test)
print(accuracy_score(Y_test, pred))
```

```
0.4482758620689655
```

```
get_accuracy('BernoulliNB SMOTE',accuracy_score(Y_test, pred))
```

```
                          Model  accuracy
0       RandomForestClassifier  0.400000
1  RandomForestClassifier GloVe  0.484375
2       DecisionTreeClassifier  0.382812
3                   BernoulliNB  0.414062
4            BaggingClassifier  0.375000
5  RandomForestClassifier SMOTE  0.413793
6  DecisionTreeClassifier SMOTE  0.321839
7          SGDClassifier SMOTE  0.344828
```

7.20 Apply BaggingClassifier using SMOTE

```
from sklearn.ensemble import BaggingClassifier
bgcl1=BaggingClassifier()
bgcl1.fit(x_train,y_train)
bgcl1.score(X_test,Y_test)
```

```
0.40229885057471265
```

```
get_accuracy('BaggingClassifier SMOTE',bgcl1.score(X_test,Y_test))
```

```
                          Model  accuracy
0       RandomForestClassifier  0.400000
1  RandomForestClassifier GloVe  0.484375
2       DecisionTreeClassifier  0.382812
3                   BernoulliNB  0.414062
4            BaggingClassifier  0.375000
5  RandomForestClassifier SMOTE  0.413793
6  DecisionTreeClassifier SMOTE  0.321839
7          SGDClassifier SMOTE  0.344828
8            BernoulliNB SMOTE  0.448276
```

```
pred = bgcl1.predict(X_test)
print(classification_report(Y_test, pred))
```

```
              precision    recall  f1-score   support

           1       0.42      0.50      0.45        10
           2       0.44      0.61      0.51        18
           3       0.30      0.22      0.26        27
           4       0.40      0.33      0.36        24
           5       0.43      0.50      0.46         6
           6       0.67      1.00      0.80         2

    accuracy                           0.40        87
   macro avg       0.44      0.53      0.47        87
weighted avg       0.39      0.40      0.39        87
```

Inference-

We applied different ML models like RandomForestClassifier, DecisionTreeClassifier, BernoulliNB, BaggingClassifier on raw data and found that Random Forest Glove give better testing accuracy than compared to other models. In order to improve our model validation, we tried replicating the data for and applied upsampling using SMOTE as there was only one value for severiety level 6 of Potential Accident Level which when splitted into train and test would get included in either of the two so to avoid this bias, the records were balanced using SMOTE and then models were applied and we observe that BernoulliNB performs far better than any other models applied so far.

Note: Although SMOTE has proved to be an effective tool for handling the class imbalance problem, it may overgeneralize the minority class as it does not take care of the distribution of majority class neighbors, especially when the minority class is very sparse with respect to the majority class. Hence in this case study we have first replicated the data for each class then proceeded with SMOTE.

8. Neural Network

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 100, kernel_initializer = 'uniform', activation = 'relu', input_dim = 142))

# Adding the second hidden layer
classifier.add(Dense(units = 200, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the second hidden layer
classifier.add(Dense(units = 50, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'softmax'))
```

```
train_y = pd.get_dummies(y_train).values
```

```
classifier.compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

```
batch_size = 20
classifier.fit(x_train, train_y, epochs = 50, batch_size=batch_size, verbose = 1)
```

```
Epoch 1/50
36/36 [==============================] - 1s 4ms/step - loss: 1.6019 - accuracy: 0.3137
Epoch 2/50
36/36 [==============================] - 0s 3ms/step - loss: 1.1871 - accuracy: 0.4762
Epoch 3/50
36/36 [==============================] - 0s 3ms/step - loss: 1.0130 - accuracy: 0.5630
Epoch 4/50
36/36 [==============================] - 0s 3ms/step - loss: 0.9320 - accuracy: 0.5854
Epoch 5/50
36/36 [==============================] - 0s 3ms/step - loss: 0.8790 - accuracy: 0.6317
Epoch 6/50
36/36 [==============================] - 0s 3ms/step - loss: 0.8433 - accuracy: 0.6457
```

```
Epoch 41/50
36/36 [==============================] - 0s 2ms/step - loss: 0.1634 - accuracy: 0.9426
Epoch 42/50
36/36 [==============================] - 0s 3ms/step - loss: 0.1315 - accuracy: 0.9566
Epoch 43/50
36/36 [==============================] - 0s 3ms/step - loss: 0.1318 - accuracy: 0.9510
Epoch 44/50
36/36 [==============================] - 0s 3ms/step - loss: 0.1149 - accuracy: 0.9636
Epoch 45/50
36/36 [==============================] - 0s 2ms/step - loss: 0.0974 - accuracy: 0.9706
Epoch 46/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0978 - accuracy: 0.9762
Epoch 47/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0789 - accuracy: 0.9748
Epoch 48/50
36/36 [==============================] - 0s 2ms/step - loss: 0.0769 - accuracy: 0.9818
Epoch 49/50
36/36 [==============================] - 0s 2ms/step - loss: 0.0642 - accuracy: 0.9818
Epoch 50/50
36/36 [==============================] - 0s 3ms/step - loss: 0.0575 - accuracy: 0.9846
```

```
get_accuracy('Neural Network',x)
```

	Model	accuracy
0	RandomForestClassifier	0.400000
1	RandomForestClassifier GloVe	0.484375
2	DecisionTreeClassifier	0.382812
3	BernoulliNB	0.414062
4	BaggingClassifier	0.375000
5	RandomForestClassifier SMOTE	0.413793
6	DecisionTreeClassifier SMOTE	0.321839
7	SGDClassifier SMOTE	0.344828
8	BernoulliNB SMOTE	0.448276
9	BaggingClassifier SMOTE	0.402299
10	Neural Network	0.268125

```
print(classification_report(Y_test, y_pred.argmax(axis=1)))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.12      0.20      0.15        10
           2       0.41      0.50      0.45        18
           3       0.41      0.48      0.44        27
           4       0.12      0.04      0.06        24
           5       0.00      0.00      0.00         6
           6       0.00      0.00      0.00         2

    accuracy                           0.29        87
   macro avg       0.15      0.17      0.16        87
weighted avg       0.26      0.29      0.26        87
```

```
t
```

	Model	accuracy
0	RandomForestClassifier	0.400000
1	RandomForestClassifier GloVe	0.484375
2	DecisionTreeClassifier	0.382812
3	BernoulliNB	0.414062
4	BaggingClassifier	0.375000
5	RandomForestClassifier SMOTE	0.413793
6	DecisionTreeClassifier SMOTE	0.321839
7	SGDClassifier SMOTE	0.344828
9	BaggingClassifier SMOTE	0.402299
10	Neural Network	0.268125

click to expand output; double click to hide output

8.2 Tunning the Neural Network Model.

```
from tensorflow.keras import regularizers
from keras.layers import BatchNormalization, Dropout
from tensorflow.keras.layers import Activation
from tensorflow.keras import optimizers
def mlp_model():
  model = Sequential()

  model.add(Dense(256, input_dim=142, kernel_initializer='he_normal'))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Dropout(0.2))
  model.add(Dense(256, kernel_initializer='he_normal'))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Dropout(0.2))
  model.add(Dense(256, kernel_initializer='he_normal'))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Dropout(0.2))
  model.add(Dense(256, kernel_initializer='he_normal'))
```

```
  model.add(Dropout(0.2))
  model.add(Dense(256, kernel_initializer='he_normal'))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Dropout(0.2))
  model.add(Dense(6, kernel_initializer='he_normal',kernel_regularizer = regularizers.l2(0.0005650011028
  model.add(Activation('softmax'))

  adam = optimizers.Adam(lr =0.01714453879813377)
  model.compile(optimizer = adam, loss = 'categorical_crossentropy', metrics = ['accuracy'])

  return model
```

```
model = mlp_model()
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 256)               36608

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 activation (Activation)     (None, 256)               0

 dropout (Dropout)           (None, 256)               0

 dense_5 (Dense)             (None, 256)               65792

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dropout_2 (Dropout)         (None, 256)               0

 dense_7 (Dense)             (None, 256)               65792

 batch_normalization_3 (Batc  (None, 256)              1024
 hNormalization)

 activation_3 (Activation)   (None, 256)               0

 dropout_3 (Dropout)         (None, 256)               0

 dense_8 (Dense)             (None, 6)                 1542

 activation_4 (Activation)   (None, 6)                 0

=================================================================
Total params: 239,622
Trainable params: 237,574
Non-trainable params: 2,048
```

```
history = model.fit(x_train, train_y, epochs = 100, verbose=1)
```

```
Epoch 1/100
23/23 [==============================] - 2s 8ms/step - loss: 1.2014 - accuracy: 0.5658
Epoch 2/100
23/23 [==============================] - 0s 8ms/step - loss: 0.7357 - accuracy: 0.7241
Epoch 3/100
23/23 [==============================] - 0s 10ms/step - loss: 0.5181 - accuracy: 0.8123
Epoch 4/100
23/23 [==============================] - 0s 9ms/step - loss: 0.4991 - accuracy: 0.8249
Epoch 5/100
23/23 [==============================] - 0s 9ms/step - loss: 0.3476 - accuracy: 0.8810
Epoch 6/100
23/23 [==============================] - 0s 8ms/step - loss: 0.3019 - accuracy: 0.8866
Epoch 7/100
23/23 [==============================] - 0s 8ms/step - loss: 0.3146 - accuracy: 0.9048
Epoch 8/100
23/23 [==============================] - 0s 9ms/step - loss: 0.2612 - accuracy: 0.9188
Epoch 9/100
23/23 [==============================] - 0s 8ms/step - loss: 0.1633 - accuracy: 0.9398
```

```
y_pred = model.predict(X_test)
```

```
3/3 [==============================] - 0s 6ms/step
```

```
x,_=model.evaluate(X_test,y_pred)
```

```
3/3 [==============================] - 0s 6ms/step - loss: 0.2329 - accuracy: 1.0000
```

```
get_accuracy('Neural Network tune',x)
```

```
                         Model  accuracy
0         RandomForestClassifier  0.400000
1   RandomForestClassifier GloVe  0.484375
2         DecisionTreeClassifier  0.382812
3                     BernoulliNB  0.414062
4              BaggingClassifier  0.375000
5   RandomForestClassifier SMOTE  0.413793
6   DecisionTreeClassifier SMOTE  0.321839
7            SGDClassifier SMOTE  0.344828
8              BernoulliNB SMOTE  0.448276
```

	Model	accuracy
0	RandomForestClassifier	0.400000
1	RandomForestClassifier GloVe	0.484375
2	DecisionTreeClassifier	0.382812
3	BernoulliNB	0.414062
4	BaggingClassifier	0.375000
5	RandomForestClassifier SMOTE	0.413793
6	DecisionTreeClassifier SMOTE	0.321839
7	SGDClassifier SMOTE	0.344828
8	BernoulliNB SMOTE	0.448276
9	BaggingClassifier SMOTE	0.402299
10	Neural Network	0.268125
11	Neural Network tune	0.232892

Inference:

Applied Dense NN with different activation functions and parameter tuning, it yields good accuracy with train data but worst validations with test data due to overfitting which might be caused due to complexity of the model but even if dropout layer were included, model didn't perform better.

We clearly see that BernoulliNB using SMOTE outperforms other models on SMOTE category and also gives better results in terms of accuracy,precision, recall and F1 measure. **Overall Random Forest Classifier Glove gives best accuracy of 48.43%**

Step 2: Design, train and test RNN or LSTM classifiers

Bidirectional recurrent neural networks(RNN) are really just putting two independent RNNs together. This structure allows the networks to have both backward and forward information about the sequence at every time step.

9.1 Checking 3 rows from the dataset

```
#list 3 rows from the dataset
df.head(3)
```

	Unnamed: 0	Data	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Genre	Employee or Third Party	Critical Risk	Description
0	0	2016-01-01 00:00:00	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
1	1	2016-01-02 00:00:00	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
2	2	2016-01-06 00:00:00	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...

9.2 dividing Date column in Day, Month and Year

```
year =[]
month =[]
date=[]
for x in range(df.shape[0]):
    h = df['Data'][x].split()
    k = h[0].split('-')
    year.append(int(k[0]))
    month.append(int(k[1]))
    date.append(int(k[2]))
```

9.3 Creating a DataFrame of Dates which will be concated with main DataFrame

```
dates = list(zip(year,month,date))
df_date = pd.DataFrame(dates, columns = ['Year','Month','Date'])
df_date
```

	Year	Month	Date
0	2016	1	1
1	2016	1	2
2	2016	1	6
3	2016	1	8
4	2016	1	10
...	...	...	...
420	2017	7	4
421	2017	7	4
422	2017	7	5

9.4 Converting Potential accident level to numerical format.

```
Potential_Accident_Level = {
    'I'   :   1,
    'II'  :   2,
    'III' :   3,
    'IV'  :   4,
    'V'   :   5,
    'VI'  :   6
}
```

9.5 Converting Months to categorical format

```
Month_Modified = {

    1   :  'January',
    2   :  'February',
    3   :  'March',
    4   :  'April',
    5   :  'May',
    6   :  'June',
    7   :  'July',
    8   :  'August',
    9   :  'September',
    10  :  'October',
    11  :  'November',
    12  :  'December'

}
```

Converting the Month Names and Potential Accident levels and dropping the original columns.

9.6 Mapping Potential Accident Level and Months to new dataframe

```
df_new['Potential_Accident_Level'] = df_new['Potential Accident Level'].map(Potential_Accident_Level)
df_new['Month_Modified'] = df_new['Month'].map(Month_Modified)
```

```
df_new.drop(['Potential Accident Level','Month'],axis=1,inplace=True)
```

Converting the Day and Year into string because we will cocatenate them with the description

```
# converting to string variables
df_new['Year'] = df_new['Year'].astype(str)
df_new['Date'] = df_new['Date'].astype(str)
```

9.7 Extracting the Y Column and dropping it from our main DataFrame

```
# Taking y Label for target
y=df_new['Potential_Accident_Level']
df_new.drop('Potential_Accident_Level', axis=1, inplace=True)
```

In the below cell, we are just adding an extra SPACE at the end of each value of a cell in a row. This will be useful when at the end we will club all these values of cells of a row together. There is no immediate use of this. But when we will sum all the values of dataframes, this will play an important role as it will put space after each word

```
for x in df_new.columns:
    df_new[x] = df_new[x] + ' '
```

First, we will separate only the description part of our main Dataframe, we will clean it. Then we will add all other columns to this column to form a single column description or our X value.

```
df_new['Description']
```

```
0      While removing the drill rod of the Jumbo 08 f...
1      During the activation of a sodium sulphide pum...
2      In the sub-station MILPO located at level +170...
3      Being 9:45 am. approximately in the Nv. 1880 C...
4      Approximately at 11:45 a.m. in circumstances t...
                             ...
420    Being approximately 5:00 a.m. approximately, w...
421    The collaborator moved from the infrastructure...
422    During the environmental monitoring activity i...
423    The Employee performed the activity of strippi...
424    At 10:00 a.m., when the assistant cleaned the ...
Name: Description, Length: 425, dtype: object
```

10. Preparing a function to clean the tweets or input string

Note:- The output of this function will be the a list of words as the sentence will be broken down after cleaning. So we will have to join those words to form the sentence

11.1 Cleaning up the data,removing stopwords,remove punctuation, converting them into tokens

```
import string
import re
import os
import nltk

from nltk.tokenize import TweetTokenizer
from nltk.corpus import stopwords, twitter_samples

tweet_tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True, reduce_len=True)

stopwords_english = stopwords.words('english')


def process_tweet(tweet):
    '''
    Input:
        tweet: a string containing a tweet
    Output:
        tweets_clean: a list of words containing the processed tweet
```

```
    #remove numbers
    tweet = re.sub(r'\d+', '', tweet)
    # remove old style retweet text "RT"
    tweet = re.sub(r'^RT[\s]+', '', tweet)
    # remove hyperlinks
    tweet = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet)
    # remove hashtags
    # only removing the hash # sign from the word
    tweet = re.sub(r'#', '', tweet)
    # tokenize tweets
    tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True, reduce_len=True)
    tweet_tokens = tokenizer.tokenize(tweet)
    ### START CODE HERE ###
    tweets_clean = []
    for word in tweet_tokens:
        if (word not in stopwords_english and # remove stopwords
            word not in string.punctuation): # remove punctuation
            #tweets_clean.append(word)
            tweets_clean.append(word)
    ### END CODE HERE ###
    return tweets_clean
```

11.2 Joining the words to form a sentence

```
desc_list1 = []
for x in desc_list:
    desc_list1.append(' '.join(x))
```

```
desc_list1[0]
```

```
'removing drill rod jumbo maintenance supervisor proceeds loosen support intermediate centralizer faci
litate removal seeing mechanic supports one end drill equipment pull hands bar accelerate removal mome
nt bar slides point support tightens fingers mechanic drilling bar beam jumbo'
```

11.3 Converting the above list into DataFrame

```
df1 = pd.DataFrame({'New_Description':desc_list1})
```

```
df1
```

	New_Description
0	removing drill rod jumbo maintenance superviso...
1	activation sodium sulphide pump piping uncoupl...
2	sub-station milpo located level collaborator e...
3	approximately nv cx ob personnel begins task u...
4	approximately circumstances mechanics anthony ...
...	...
420	approximately approximately lifting kelly hq t...
421	collaborator moved infrastructure office julio...
422	environmental monitoring activity area employe...

Important thing to note here is that the above dataframe is just the description part of our main DataFrame. So, we need to concat the above DataFrame into our main dataframe, then add all the columns to form one single description.

```
df1 = pd.concat([df_new,df1],axis=1)
```

```
df1.head(2)
```

	Countries	Local	Industry Sector	Genre	Employee or Third Party	Critical Risk	Description	Year	Date	Month_Modified	New_Description
0	Country_01	Local_01	Mining	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...	2016	1	January	removing drill rod jumbo maintenance superviso...
1	Country_02	Local_02	Mining	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	2	January	activation sodium sulphide pump piping uncoupl...

```
df1.drop('Description',axis=1,inplace=True)
```

```
df1.head(2)
```

	Countries	Local	Industry Sector	Genre	Employee or Third Party	Critical Risk	Year	Date	Month_Modified	New_Description
0	Country_01	Local_01	Mining	Male	Third Party	Pressed	2016	1	January	removing drill rod jumbo maintenance superviso...
1	Country_02	Local_02	Mining	Male	Employee	Pressurized Systems	2016	2	January	activation sodium sulphide pump piping uncoupl...

Note: Here, we are adding all the values of a dataframe and storing it in the column 'Description'

```
df1['Description'] = df1.sum(axis=1)
df1.head(2)
```

	Countries	Local	Industry Sector	Genre	Employee or Third Party	Critical Risk	Year	Date	Month_Modified	New_Description	Description
0	Country_01	Local_01	Mining	Male	Third Party	Pressed	2016	1	January	removing drill rod jumbo maintenance superviso...	Country_01 Local_01 Mining Male Third Party Pr...
1	Country_02	Local_02	Mining	Male	Employee	Pressurized Systems	2016	2	January	activation sodium sulphide pump piping uncoupl...	Country_02 Local_02 Mining Male Employee Press...

Going forward, this forms our X value. We will convert these words to tensor and feed to the LSTM Model

Methodology

Post preprocessing, all the columns were clubbed together with Description

11.4 Getting all the columns in description

```
df1['Description'][0]
```

```
'Country_01 Local_01 Mining Male Third Party Pressed 2016 1 January removing drill rod jumbo maintenan
ce supervisor proceeds loosen support intermediate centralizer facilitate removal seeing mechanic supp
orts one end drill equipment pull hands bar accelerate removal moment bar slides point support tighten
s fingers mechanic drilling bar beam jumbo'
```

11. Extracting total Vocabulary words

11.1 Creating vocabulary for the words and passing tokens to each word

```
# Build the vocabulary
# Unit Test Note - There is no test set here only train/val

# Include special tokens
# started with pad, end of line and unk tokens
Vocab = {'__PAD__': 0, '____': 1, '__UNK__': 2}

# Note that we build vocab using training data
for sentence in df1['Description']:
    for word in sentence.split():
        if word not in Vocab:
            Vocab[word] = len(Vocab)

print("Total words in vocab are",len(Vocab))
display(Vocab)
```

```
Total words in vocab are 3173

{'__PAD__': 0,
 '____': 1,
 '__UNK__': 2,
 'Country_01': 3,
 'Local_01': 4,
 'Mining': 5,
 'Male': 6,
 'Third': 7,
 'Party': 8,
 'Pressed': 9,
 '2016': 10,
 '1': 11,
 'January': 12,
 'removing': 13,
 'drill': 14,
 'rod': 15,
 'jumbo': 16,
```

Indexing each word for tensors value. When processing sequence data, it is very common for individual samples to have different lengths, hence used padding. Padding comes from the need to encode sequence data into contiguous batches: in order to make all sequences in a batch fit a given standard length, it is necessary to pad or truncate some sequences. Since the input data for a deep learning model must be a single tensor , samples that are shorter than the longest item need to be padded with some placeholder value.

11.2 Writing a function to convert the description to tensor

```
def tweet_to_tensor(tweet, vocab_dict, unk_token='__UNK__', verbose=False):
    '''
    Input:
        tweet - A string containing a tweet
        vocab_dict - The words dictionary
        unk_token - The special string for unknown tokens
        verbose - Print info durign runtime
    Output:
        tensor_l - A python list with

    '''

    ### START CODE HERE (Replace instances of 'None' with your code) ###
    # Process the tweet into a list of words
    # where only important words are kept (stop words removed)
    word_l = tweet.split()

    if verbose:
        print("List of words from the processed tweet:")
        print(word_l)
```

```
    if verbose:
        print(f"The unique integer ID for the unk_token is {unk_ID}")

    # for each word in the list:
    for word in word_l:

        # Get the unique integer ID.
        # If the word doesn't exist in the vocab dictionary,
        # use the unique ID for __UNK__ instead.
        word_ID = Vocab[word] if word in Vocab else unk_ID
    ### END CODE HERE ###

        # Append the unique integer ID to the tensor list.
        tensor_l.append(word_ID)

    return tensor_l
```

Tensors Created

```
tensor_values=[]

for row in df1['Description']:
          tensor_values.append(tweet_to_tensor(row, Vocab, unk_token='__UNK__', verbose=False))
```

```
df1['Description'][99]
```

```
'Country_02 Local_05 Metals Male Third Party Pressurized Systems / Chemical Substances 2016 22 April e
mployee reports placed air lance tank opened manual air valve projection acid solution heated toward r
eaching front left thigh'
```

```
y[99]
```

```
4
```

11.3 Printing out description column and their appriopriate Tokens alloacation

```
np.array(tensor_values[5])
```

```
array([ 43, 182, 183,   6,   7,   8,  75,  46,  47,  10, 184,  12, 185,
       186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198,
       199, 200, 201, 202, 192, 203, 187, 204,  83, 205, 206, 207])
```

11.4 Maximum length of the Sentence

```
max_len = 0

for x in range(len(tensor_values)):
    a = len(tensor_values[x])
    if a > max_len:
        max_len = a

max_len
```

```
105
```

11. Model building Bi-LSTM

11.1 Padding the X values

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
maxlen = 100
X = pad_sequences(maxlen=maxlen, sequences=tensor_values, padding="post", value=0)
```

11.2 Building Bidirectional LSTM Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM, SpatialDropout1D,Bidirectional
from tensorflow.keras import regularizers
from tensorflow.keras import optimizers

embed_dim = 256
lstm_out = 64

model = Sequential()
model.add(Embedding(len(Vocab), embed_dim,input_length = X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(Bidirectional(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2)))
model.add(Dense(5,activation='softmax',kernel_regularizer = regularizers.l2(0.0015650011028133052)))
adam = optimizers.Adam(lr =0.003315453879813377)
model.compile(optimizer=adam,loss = 'categorical_crossentropy',metrics = ['accuracy'])
print(model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 256)          812288

 spatial_dropout1d (SpatialD  (None, 100, 256)         0
 ropout1D)

 bidirectional (Bidirectiona  (None, 128)              164352
 l)

 dense_9 (Dense)             (None, 5)                 645

=================================================================
Total params: 977,285
Trainable params: 977,285
Non-trainable params: 0
```

12. Divide the data into training and testing set

12.1 Creating Y categorical Data and splitting the data

```
from sklearn.model_selection import train_test_split

Y = pd.get_dummies(y).values
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.30, random_state = 33)


print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

```
(297, 100) (297, 6)
(128, 100) (128, 6)
```

Droping out Accident Level 6 as it only has 1 data point. If we spilt into train and testing set level 6 Accident level will either fit into training or testing so it wont be any use. Also while using SMOTE we atleast require the number of class present in the target column i.e we had 6 classes in target column so we require atleast 6 numbers of rows in each class of Accident Level.

```
k = pd.DataFrame(Y_train)
k.drop(5,inplace=True,axis=1)
```

```
Y_train = np.asarray(k)
```

Oversampling of data because of data imbalance. Using SMOTE to do the oversampling

```
from imblearn.over_sampling import SMOTE
oversample = SMOTE()

X_ROS, y_ROS = oversample.fit_resample(X_train, Y_train)
print(X_ROS.shape,'\n',y_ROS.shape)
#k=pd.DataFrame(y_ROS)
```

```
(545, 100)
 (545, 5)
```

```
y_ROS.shape
```

```
(545, 5)
```

```
X_ROS_, y_ROS_ = oversample.fit_resample(X_test, Y_test)
print(X_ROS_.shape,'\n',y_ROS_.shape)
```

```
(195, 100)
 (195, 5)
```

```
batch_size = 20
model.fit(X_ROS, y_ROS, epochs = 20,validation_data =(X_ROS_,y_ROS_) , batch_size=batch_size, verbose =
```

```
Epoch 1/20
28/28 [==============================] - 18s 398ms/step - loss: 1.5072 - accuracy: 0.3266 - val_los
s: 1.4152 - val_accuracy: 0.4154
Epoch 2/20
28/28 [==============================] - 10s 371ms/step - loss: 1.0684 - accuracy: 0.5853 - val_los
s: 2.6754 - val_accuracy: 0.4205
Epoch 18/20
28/28 [==============================] - 10s 370ms/step - loss: 0.0452 - accuracy: 0.9945 - val_los
s: 2.8178 - val_accuracy: 0.4308
Epoch 19/20
28/28 [==============================] - 10s 368ms/step - loss: 0.0354 - accuracy: 0.9963 - val_los
s: 2.8844 - val_accuracy: 0.4154
Epoch 20/20
28/28 [==============================] - 10s 371ms/step - loss: 0.0301 - accuracy: 0.9982 - val_los
s: 2.9426 - val_accuracy: 0.4205
```

```
<keras.callbacks.History at 0x7f2dd63f89d0>
```

```
score,acc = model.evaluate(X_ROS_, y_ROS_, verbose = 2, batch_size = batch_size)
print("loss: %.2f" % (score))
print("acc: %.2f" % (acc))
```

```
10/10 - 0s - loss: 2.9426 - accuracy: 0.4205 - 391ms/epoch - 39ms/step
loss: 2.94
acc: 0.42
```

12.2 Classification Report

```
from sklearn.metrics import classification_report

pred1 = model.predict(X_ROS_)

print(classification_report(y_ROS_.argmax(axis=1), pred1.argmax(axis=1)))
```

```
7/7 [==============================] - 1s 56ms/step
              precision    recall  f1-score   support

           0       0.63      0.49      0.55        39
           1       0.30      0.38      0.34        39
           2       0.36      0.38      0.37        39
           3       0.41      0.38      0.39        39
           4       0.50      0.46      0.48        39

    accuracy                           0.42       195
   macro avg       0.44      0.42      0.43       195
weighted avg       0.44      0.42      0.43       195
```

12.3 Confusion Matrix

```
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt

print(model.evaluate(X_ROS_, y_ROS_))
y_predict = model.predict(X_ROS_)

cm = metrics.confusion_matrix(y_ROS_.argmax(axis=1), pred1.argmax(axis=1), labels = [0,1,2,3,4])

df_cm = pd.DataFrame(cm , index = [i for i in ["I", "II","III","IV","V"]],
                     columns = [i for i in ["I","II","III","IV","V"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True, fmt='g')
```

```
7/7 [==============================] - 0s 53ms/step - loss: 2.9426 - accuracy: 0.4205
[2.942645788192749, 0.4205128252506256]
7/7 [==============================] - 0s 51ms/step
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2ddba6cf40>
```

```
7/7 [==============================] - 0s 53ms/step - loss: 2.9426 - accuracy: 0.4205
[2.942645788192749, 0.4205128252506256]
7/7 [==============================] - 0s 51ms/step

matplotlib.axes._subplots.AxesSubplot at 0x7f2ddba6cf40>
```

This state that we have classified:

1. Around 58% of class I Accidents
2. 36 % of class II Accidents
3. 32 % of class III Accidents
4. 33 % of class IV Accidents
5. 62 % of class V Accidents

12.4 Due to insufficiant data points we see less Accuracy in the report.

```
acct = []
mod = []
def get_accuracy(y,x):
    global t
    acct.append(x)
    mod.append(y)
    temp=pd.DataFrame(mod,columns=['Model'])
    temp1 = pd.DataFrame(acct,columns=['accuracy'])
    t =temp.join(temp1)
    print(t)
```

```
ace = pd.read_csv('Modelaccuracy.csv')
```

```
acc
```

```
0.4205128252506256
```

```
get_accuracy('LSTM NN',acc)
```

```
      Model  accuracy
  0  LSTM NN  0.420513
```

```
t
```

	Model	accuracy
0	LSTM NN	0.420513

```
ace.drop('Unnamed: 0',axis=1,inplace=True)
```

```
_ace=ace.append(t)
```

```
_ace.shape
```

```
(13, 2)
```

```
_ace
```

	Model	accuracy
0	RandomForestClassifier	0.400000
1	RandomForestClassifier GloVe	0.484375
2	DecisionTreeClassifier	0.382812
3	BernoulliNB	0.414062
4	BaggingClassifier	0.375000
5	RandomForestClassifier SMOTE	0.413793
6	DecisionTreeClassifier SMOTE	0.321839
7	SGDClassifier SMOTE	0.344828
8	BernoulliNB SMOTE	0.448276
9	BaggingClassifier SMOTE	0.402299
10	Neural Network	0.268125

12.5 Pickling or Saving the Model

```
model.save('model.h5')
```

13. Milestone 3: [Optional]

Step 1: Design a clickable UI based chatbot interface

We will be exploring Tkinter – python GUI programming tool to build our chatbot for this project. We will explore how we can deploy a model and check real-time predictions using Tkinter. We will first build a classification model that will classify on Potential Accident severity level. Then we will make a GUI using Tkinter and will check predictions on new data points.

Methodology-

[1]Creating GUI

Tkinter is a library written in Python that is widely used to create GUI applications. It is very easy to build GUI using Tkinter and the process is even faster. Tkinter has several widgets that can be used while developing GUI. These include buttons, radio buttons, checkboxes, etc. We will see how we can make a GUI Tkinter after we build the model.

[2]Model Building

As discussed earlier, we are proceeding ahead with Bidirectional LSTM model as the accuracy is higher than compared to other models. Once we are done with the model validation, we pickle this model that would be used to compute predictions for new data points.

[3]Computing Predictions Using the Tkinter GUI in Real-Time

To predict the class, we will need to provide input in the same way as we did while training. So we will create some functions that will perform text preprocessing and then predict the class. After predicting the class, we will get a random response from the list of intents. Now we will develop a graphical user interface. We will take the input message from the user and then use the helper functions we have created to get the response from the bot and display it on the GUI

13.1 Importing necessary libraries

```
#Importing necessary libraries
import nltk
nltk.download('punkt')
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
import json
import pickle

import numpy as np
import random
import string
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\jiten\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
```

13.2. Creating intents and checking values of the inputs

```
intents ={
         "Intro":{
         "patterns": ["hi",
                      "how are you",
                      "is anyone there",
                      "hello",
                      "whats up",
                      "hey",
                      "yo",
                      "listen",
                      "please help me",
                      "i am learner from",
                      "i belong to",
                      "aiml batch",
                      "aifl batch",
                      "i am from",
                      "my pm is",
                      "blended",
                      "online",
                      "i am from",
                      "hey ya",
                      "talking to you for first time"],
         "responses": ["Hello! how can i help you ?"],
         "context_set": ""
        },
        "Exit" :{
         "patterns": ["thank you",
                      "thanks",
                      "cya",
                      "see you",
                      "later",
                      "see you later",
                      "goodbye",
                      "i am leaving",
                      "have a Good day",
                      "you helped me",
                      "thanks a lot",
                      "thanks a ton",
                      "you are the best",
                      "great help",
                      "too good",
```

```
    "Party" :{'Third Party', 'Employee', 'Third Party (Remote)'},

    "Critical_Risk" :{'Pressed', 'Pressurized Systems', 'Manual Tools', 'Others',
       'Fall prevention (same level)', 'Chemical substances',
       'Liquid Metal', 'Electrical installation', 'Confined space',
       'Pressurized Systems / Chemical Substances',
       'Blocking and isolation of energies', 'Suspended Loads', 'Poll',
       'Cut', 'Fall', 'Bees', 'Fall prevention', '\nNot applicable',
       'Traffic', 'Projection', 'Venomous Animals', 'Plates',
       'Projection/Burning', 'remains of choco',
       'Vehicles and Mobile Equipment', 'Projection/Choco',
       'Machine Protection', 'Power lock', 'Burn',
       'Projection/Manual Tools', 'Individual protection equipment',
       'Electrical Shock', 'Projection of fragments'},



}
```

13.3 Initializing the variables to be used to store values from Chats

```
class chatBot:
    def __init__(self,Day=1,Month='null',Year=0,Country='',Locality='',Industry='',Gender='',PartyPerson
        self._Day = Day
        self._Month = Month
        self._Year = Year
        self._Country = Country
        self._Locality = Locality
        self._Industry = Industry
        self._Gender = Gender
        self._PartyPerson = PartyPerson
        self._CriticalRisk = CriticalRisk
        self._Description = Description
    #Getter Method
    def get(self):
        return self._Day,self._Month,self._Year

    def setFunc(self,a):
           self._Day = a
```

```
    def setFunc(self,a):
           self._Day = a

    def setMon(self,x):
          self._Month = x

    def setYear(self,b):
          self._Year  = b
    def setCountry(self,c):
            self._Country  = c
    def setLocality(self,d):
            self._Locality  = d
    def setIndustry(self,e):
            self._Industry = e
    def setGender(self,f):
            self._Gender  = f
    def setPartyPerson(self,g):
            self._PartyPerson = g
    def setCriticalRisk(self,h):
            self._CriticalRisk  = h
```

14.4 Loading the pickled Model

```
from tensorflow.keras import models
model = models.load_model('model.h5')
```

13.4Creating the function to use the inputs taken from the chat to make the prediction

```
def prediction():
    list = [p1._Country, p1._Locality, p1._Industry, p1._Gender, p1._PartyPerson, p1._CriticalRisk, p1._
    Data =''
    for x in list:
        Data += x
        Data += ' '
    from utils import tweet_to_tensor

    tensor_x = tweet_to_tensor(Data, 3173, unk_token='__UNK__', verbose=False)

    from tensorflow.keras.preprocessing.sequence import pad_sequences
    maxlen = 100

    padding = [0]*(maxlen - len(tensor_x))

    X = tensor_x + padding
    X = np.array(X)
    X.shape = (1,100)
```

14.5 Creating the function to ask question and take the User Inputs

```
#Creating GUI with tkinter
import tkinter
from tkinter import *


def send():
    msg = EntryBox.get("1.0",'end-1c').strip()
    EntryBox.delete("0.0",END)

    if msg != '':
        ChatLog.config(state=NORMAL)
        ChatLog.insert(END, "You: " + msg + '\n\n')
        ChatLog.config(foreground="#442265", font=("Verdana", 12 ))

        if msg in intents['Intro']["patterns"]:
            ChatLog.insert(END, "Bot: " + str(random.sample(intents['Intro']["responses"],1)[0]) + '\n\
            #ChatLog.config(state=DISABLED)
        elif msg in intents['Bot']["patterns"]:
```

```
            ChatLog.insert(END, "Bot: " + 'Please Describe in Detail' + '\n\n')

            p1.setCriticalRisk(msg)

        elif len(msg)>30:

            p1.setDescription(msg)
            x=prediction()
            ChatLog.insert(END, "Bot: " + 'The Potential Accident Level is' + ' ' + str(x+1) + '\n\n')









#         res = chatbot_response(msg)
#         ChatLog.insert(END, "Bot: " + res + '\n\n')
    ChatLog.config(state=DISABLED)
```

13.5 Create virtual display

```
### CREATE VIRTUAL DISPLAY ###
#!apt-get install -y xvfb # Install X Virtual Frame Buffer
#import os
#os.system('Xvfb :1 -screen 0 1600x1200x16  &')    # create virtual display with size 1600x1200 and 16 b
#os.environ['DISPLAY']=':1.0'    # tell X clients to use our virtual DISPLAY :1.0.
```

```
base = Tk()
base.title("IBot")
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)
```

13.6 Create Chat window

```
#Create Chat window
ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",)

ChatLog.config(state=DISABLED)
```

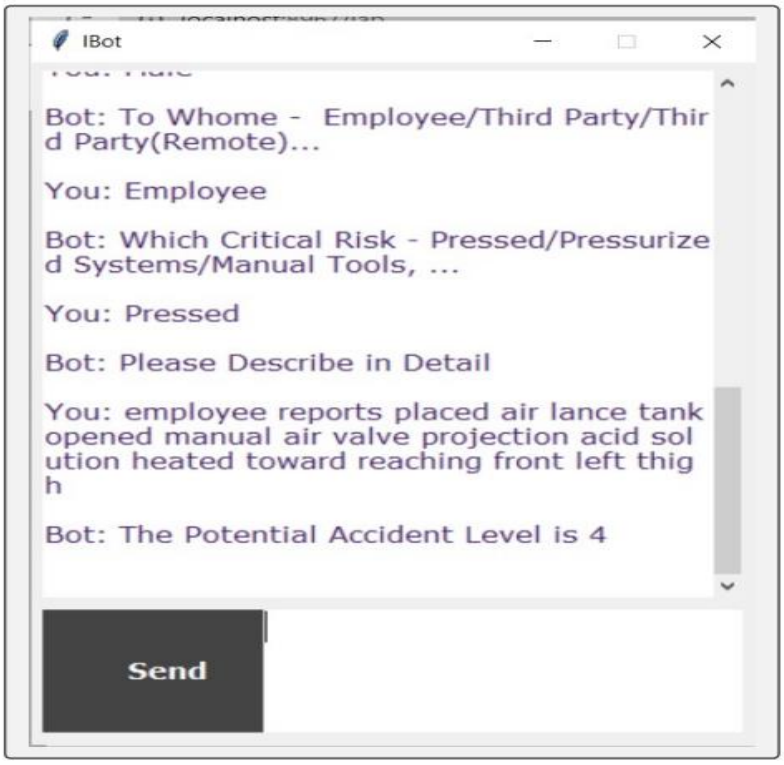
14.6 Main loop of Tkinter

```
#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")
ChatLog['yscrollcommand'] = scrollbar.set

#Create Button to send message
SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12", height=5,
                    bd=0, bg="#444444", activebackground="#5c9d9b",fg='#ffffff',
                    command= send )

#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")
#EntryBox.bind("", send)


#Place all components on the screen
scrollbar.place(x=376,y=6, height=386)
ChatLog.place(x=6,y=6, height=386, width=370)
EntryBox.place(x=128, y=401, height=90, width=265)
SendButton.place(x=6, y=401, height=90)
```

```
base.mainloop()
```

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\jiten\anaconda3\lib\tkinter\__init__.py", line 1892, in __call__
    return self.func(*args)
  File "C:\Users\jiten\AppData\Local\Temp/ipykernel_3684/657645183.py", line 69, in send
    x=prediction()
  File "C:\Users\jiten\AppData\Local\Temp/ipykernel_3684/259493852.py", line 7, in prediction
    from utils import tweet_to_tensor
ModuleNotFoundError: No module named 'utils'
```

Chatbot Output Window :



Inference:

In this project, we understood about chatbots and implemented a deep learning version of a chatbot in Python. Probably we are suffering from Data Insufficiency, hence even upsampling of data is not helping much on modelling accuracy. However, we have successfully built a semi-ruled chatbot which helps predicting Potential Accident Level in accordance with the user's input.

Future Work: We can enhance this project by building AI chatbot.

References

1) https://www.chatbot.com/chatbot-guide/
2)https://www.forbes.com/sites/cognitiveworld/2020/02/23/ choosing-between-rule-based-bots-and-ai-bots/?sh=35b006f0353d
3)https://analyticsindiamag.com/complete-tutorial-on-tkinter-to-deploy-machine-learning-model/
4)https://chatbotslife.com/introducing-conversational-chat-bots-using-rule-based-approach-c8840aeaad07
5)https://www.python-engineer.com/posts/chatbot-gui-tkinter